Le résumé linguistique de données structurées comme support pour l'interrogation

W. Amenel Voglozin

École Centrale de Nantes **Université de Nantes** **École des Mines de Nantes**

ÉCOLE DOCTORALE STIM

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX »

Année 2007

Le résumé linguistique de données structurées comme support pour l'interrogation

---

THÈSE DE DOCTORAT

Discipline : INFORMATIQUE

Spécialité : INFORMATIQUE

Présentée
et soutenue publiquement par

W. Amenel Abraham VOGLOZIN

*Le 11 juillet 2007 à l'*UFR *Sciences & Techniques, Université de Nantes,*
devant le jury ci-dessous

Président	:	Pr. Mokrane BOUZEGHOUB	PRiSM, Université de Versailles
Rapporteurs	:	Daniel ROCACHER, H.D.R	IRISA, Université de Rennes 1
		Florence SÈDES, Pr.	IRIT, Université Paul Sabatier
Examinateurs	:	Guillaume RASCHIA, M.C.	LINA, Université de Nantes
		Laurent UGHETTO, M.C.	IRISA, Université de Rennes 2

Directeur de thèse : Pr. Noureddine MOUADDIB

Laboratoire: **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED 0366-311

LE RÉSUMÉ LINGUISTIQUE DE DONNÉES STRUCTURÉES COMME SUPPORT POUR L'INTERROGATION

Linguistic summaries of structured data as a tool for querying

W. Amenel Abraham VOGLOZIN

⋈

favet neptunus eunti

Université de Nantes

W. Amenel Abraham VOGLOZIN

Le résumé linguistique de données structurées comme support pour l'interrogation

IV+x+178 p.

Ce document a été préparé avec LaTeX2e et la classe `these-LINA` version v. 2.7 de l'association de jeunes chercheurs en informatique LoGiN, Université de Nantes. La classe `these-LINA` est disponible à l'adresse : `http://login.irin.sciences.univ-nantes.fr/`.

Cette classe est conforme aux recommandations du ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche (circulaire n° 05-094 du 29 mars 2005), de l'Université de Nantes, de l'école doctorale « Sciences et Technologies de l'Information et des Matériaux » (ED-STIM), et respecte les normes de l'association française de normalisation (AFNOR) suivantes :

- AFNOR NF Z41-006 (octobre 1983)
  Présentation des thèses et documents assimilés ;
- AFNOR NF Z44-005 (décembre 1987)
  Documentation – Références bibliographiques – Contenu, forme et structure ;
- AFNOR NF Z44-005-2/ISO NF 690-2 (février 1998)
  Information et documentation – Références bibliographiques – Partie 2 : documents électroniques, documents complets ou parties de documents.

Impression : These.tex – 23/09/2007 – 14:34.

Révision pour la classe : these-LINA.cls,v 2.7 2006/09/12 17:18:53 mancheron Exp

Résumé

Le travail présenté dans cette thèse traite de l'utilisation des résumés de données dans l'interrogation. Dans le contexte des résumés linguistiques du modèle SaintEtiQ sur lequel se focalise cette thèse, un résumé est une description du contenu d'une table relationnelle. Grâce à la définition de variables linguistiques, il est possible d'utiliser des termes du langage pour caractériser les données structurées de la table. En outre, l'organisation des résumés en hiérarchie offre divers niveaux de granularité. Nous nous intéressons à fournir une application concrète aux résumés déjà construits. D'une part, nous étudions les possibilités d'utilisation des résumés dans une interrogation à but descriptif. L'objectif est de décrire entièrement des données dont certaines caractéristiques sont connues. Nous proposons une démarche de recherche de concepts et une instanciation de cette démarche. Ensuite, une étude des systèmes d'interrogation flexible, dont certains ont, ainsi que SaintEtiQ, la théorie des sous-ensembles flous comme base, nous permet d'enrichir la démarche proposée par des fonctionnalités plus avancées. D'autre part, nous avons intégré les résumés linguistiques de SaintEtiQ au SGBD PostgreSQL. L'objectif est d'aider le SGBD à identifier des enregistrements. Nous présentons un état de l'art des techniques d'indexation, ainsi que le détail de l'implémentation des résumés en tant que méthode d'accès dans PostgreSQL.

Mots-clés : SaintEtiQ, résumés linguistiques, données structurées, interrogation de résumés, sous-ensembles flous, interrogation flexible, aspects coopératifs, index, techniques d'indexation, méthode d'accès, PostgreSQL.

Abstract

This thesis deals with using summaries of data in a querying process. The work discussed focuses on the linguistic summaries of the SaintEtiQ model, in which a summary describes the content of a relational table. The definition of linguistic variables allows to use terms from the natural language to characterize the structured data. In addition, the organization of summaries into a hierarchy based on generalization links offers various levels of granularity. On one hand, we study the possible use of summaries in a descriptive querying process. The aim is to characterize entirely the data on the basis of a partial characterization. We propose an approach which consists in searching for specific concepts, from the expression of queries to the presentation of results. An instantiation of the approach using a summary hierarchy traversal algorithm is part of the proposal. Then, a survey of flexible querying systems, some of which are based on fuzzy sets as SaintEtiQ is, allows us to provide additional functionalities to the querying approach. On the other hand, we integrate linguistic summaries of the model into the DBMS PostgreSQL. The aim is to help the DBMS identify tuples by exploiting a summary hierarchy as an index structure. We provide a survey of indexing techniques as well as the details of implementing summaries as an index method under PostgreSQL.

Keywords: SaintEtiQ, linguistic summaries, structured data, querying of summaries, fuzzy sets, flexible querying, cooperative aspects, index, indexing techniques, access method, PostgreSQL.

Remerciements

Mes remerciements vont aux membres du jury pour m'avoir offert temps et intérêt.

Merci à mon directeur de thèse, M. Noureddine Mouaddib, pour l'occasion de poursuivre une thèse, pour l'environnement de travail sans défaut et pour avoir suivi mon travail.

Merci à mes encadrants, Laurent Ughetto et Guillaume Raschia, pour la substance de ce travail.

Merci à Maeva d'avoir été présente.

Merci à ma famille, proche, éloignée ou de cœur et à mes amis pour leur soutien constant. Je pense notamment à Hélène C., Hélène M., Alain M. et Joëlle M.

Enfin, merci aux éducateurs que j'ai eu l'occasion de croiser et qui ont tous participé à me façonner.

Sommaire

— *Corps du document* —

— *Pages annexées* —

Introduction

Problématique, motivation et objectifs

La croissance ininterrompue des capacités de stockage semble liée aux progrès technologiques. Il fut une époque où le moindre bit était utilisé presque à contrecœur en raison de la faible disponibilité de l'espace de stockage. L'algorithme de Huffman [84] est représentatif des techniques de compression destinées à une meilleure utilisation de l'espace disponible. Pour le grand public, cette croissance est plutôt synonyme de confort : plus d'espace pour stocker ses données, moins de soucis de rangement, plus besoin de trier pour faire de la place, il suffit de passer à une capacité supérieure quand le besoin s'en fait sentir. Mais cette croissance pose un réel problème pour les traitements informatiques parce qu'elle implique une augmentation du volume de données à traiter.

Il est en effet évident qu'un traitement, même très rapide pour une unité de donnée élémentaire, est immanquablement « submergé » par le volume des données. Les exemples sont nombreux. C'est ainsi que l'encryptage et le décryptage de données par clé symétrique est suffisamment rapide pour induire un faible surcoût en temps et être effectué au vol, par exemple, l'algorithme RC4 de Ronald Rivest, est utilisé pour sécuriser les communications (protocoles SSL sur Internet et WEP dans les systèmes WiFi). Mais le décryptage par *brute-force attack*[1] ne bénéficie pas de cette efficience [132]. De même, les changements d'affectation du processeur dans un système d'exploitation multitâche préemptif donnent l'illusion de tâches simultanées sur un ordinateur monoprocesseur. Mais le phénomène de *trashing*[2] s'impose à tous les systèmes, même les plus performants.

La croissance du volume des données à traiter ouvre la voie à de nouveaux champs d'étude. Dans certains cas, il est possible d'atténuer l'influence de la quantité de données sur les temps de réponse des traitements et algorithmes. Les traitements concernés peuvent se satisfaire d'approximations, généralement de moindre qualité que des résultats non approximatifs, mais obtenues en des temps plus acceptables. C'est le cas de l'échantillonnage au sein d'une popu-

[1] Le décryptage par brute-force attack consiste à essayer toutes les clés possibles pour décrypter un texte codé. Ce type de décryptage suppose de pouvoir déterminer si le texte décrypté est susceptible d'être le texte en clair initial, inconnu.

[2] Le *trashing* désigne, en systèmes d'exploitation, une chute brutale des performances d'un système en raison d'une surcharge. En termes familiers, on dit que « le système rame ».

lation ou des problèmes NP-complets dont une solution approchée peut être obtenue par des algorithmes polynomiaux. Parmi les approches utilisées pour atténuer l'impact du volume de données, on trouve les méthodes de résumé par généralisation [26, 53, 97, 115, 150].

Un autre problème auquel sont confrontés des traitements informatiques est celui de la difficulté à modéliser des représentations « humaines » ou « naturelles ». Mentionnons en premier lieu la notion de *gradualité*, qui revêt divers sens, parmi lesquels une évolution progressive d'un état, ou le degré de satisfaction variable d'une caractéristique. La gradualité ne peut pas être reflétée par la dichotomie des systèmes binaires. Les nuances et autres gradations sont ainsi occultées, donnant lieu à un *effet de seuil*. Par exemple, la condition « il faut avoir une taille de 172 cm pour postuler » exclut la taille de 170 cm, néanmoins proche de 172 cm. De même, le caractère « jeune » d'une personne ne peut pas être nuancé dans un mode binaire : on est soit « jeune », soit « pas jeune » mais on ne peut être rien d'autre, encore moins être les deux à la fois. Ensuite, la notion de préférence, qui désigne un souhait plutôt qu'une contrainte forte, n'est pas un élément de base des SGBD, en plus d'être difficile à modéliser.

La théorie des sous-ensembles flous, proposée par Zadeh [151] est une généralisation de la théorie des ensembles. Cet outil mathématique définit le concept d'appartenance partielle à un ensemble et offre une solution aux problèmes de modélisation dûs à l'effet de seuil. Il devient ainsi possible d'exprimer un caractère graduel, phénomène courant dans le langage naturel, par exemple sur les termes descriptifs (« grand », « jeune », « léger », etc.). On exprime plus facilement des transitions graduelles, des situations intermédiaires, des informations imprécises ou des classes aux limites mal définies. La théorie des sous-ensembles flous est utilisée par toutes les méthodes de résumé qui ont été recensées. Elle permet à ces méthodes de décrire les données par des termes « linguistiques » car issus du langage naturel. Les résumés produits sont alors appelés des « résumés linguistiques ».

Le travail présenté dans ce rapport de thèse est lié à la réduction des données et à la description linguistique de données. Il traite en effet des résumés linguistiques de données structurées du modèle SAINTETIQ. Ce modèle utilise un ensemble de termes linguistiques, dont la modélisation est explicitement graduelle, pour décrire des données structurées. Il a également pour objectif de produire des « versions » condensées des données. Le résultat du processus de résumé est donc une synthèse du contenu d'une table relationnelle. C'est à ce titre que ce modèle s'inscrit dans l'optique de réduction des données, comme les autres méthodes de résumés linguistiques. Il faut cependant noter qu'une fois les résumés produits, aucune exploitation ultérieure n'en est faite dans le cas général. Nous proposons ici d'aller plus loin avec les résumés

du modèle SAINTETIQ, en étudiant l'interrogation des hiérarchies de résumés produites par le modèle.

Notre objectif est d'offrir les moyens de trouver les résumés ou les données satisfaisant des critères spécifiés. Le modèle relationnel et son langage d'interrogation, SQL, nous servant de références, le lecteur retrouvera les éléments familiers de requêtes, de conditions, de résultats, de procédure d'évaluation ou d'index. Néanmoins, le but affiché n'est pas de reproduire, au niveau des résumés de données, la richesse du langage SQL, fruit d'un travail considérable effectué par une communauté de chercheurs pendant des décennies. Notre ambition, plus modeste, fait partie d'un projet qui vise à doter les résumés SAINTETIQ d'outils d'interrogation qui manquent aux techniques de résumé de données.

Structure du document

Ce document est constitué de six chapitres organisés comme suit. Le premier chapitre a pour objectif la compréhension du modèle de résumés SAINTETIQ. Il expose d'abord les techniques de réduction de données, dont font partie les techniques de résumé. Ces techniques, dont le but est de réduire le volume de données en entrée d'un traitement, fournissent une forme d'approximation quantitative et/ou qualitative. Les techniques de réduction utilisent des méthodes qui visent à préserver les informations nécessaires aux applications finales. Ce premier chapitre situe les résumés du modèle SAINTETIQ dans ce contexte (méthodes et types d'approximation) avant de les présenter. Cette présentation porte sur la nature des données prises en compte dans la construction des résumés (données structurées et vocabulaire de description de ces données), sur les étapes de la construction, sur les représentations syntaxiques des résumés et leur propriétés et, enfin, sur les informations qu'on peut tirer de ces représentations et propriétés.

Le chapitre 2 propose une démarche d'interrogation des hiérarchies de résumés SAINTETIQ. L'interrogation est entendue comme la recherche et la présentation des résumés qui satisfont des critères fournis en entrée du processus d'interrogation. La proposition porte sur les entrées du processus, le traitement effectif et la sortie. En entrée, une représentation canonique des requêtes sous forme de conditions logiques en forme normale conjonctive est proposée. Le traitement est instancié par un algorithme d'exploration des hiérarchies, avec coupures de branches, pour lequel les tests de correspondance d'un résumé avec la requête sont détaillés. En sortie, une présentation des résumés résultats sous forme de pseudo-classes d'équivalence vis-à-vis de critères de recherche est proposée.

Le chapitre 3 présente un état de l'art de l'interrogation flexible en bases de données. Synthétisé à partir de publications consacrées au sujet, il classe les systèmes d'interrogation flexible en fonction de leurs fondements théoriques. Il montre également que la notion de « préférence » fait partie intégrante des motivations de ces systèmes. Ce chapitre présente donc d'une part, les préférences sous divers aspects généraux et, d'autre part, quelques systèmes d'interrogation flexible choisis sur la base d'une documentation disponible ou de leur intérêt historique.

Le chapitre 4 regroupe l'ensemble de nos contributions, inspirées des systèmes d'interrogation flexible, à l'interrogation des résumés SAINTETIQ. En premier lieu, il présente une implémentation de la démarche proposée au chapitre 2, pour l'interrogation approchée par généralisation de données structurées. Ensuite, il présente les principes d'une fonctionnalité coopérative rajoutée à l'interrogation des résumés. Le caractère coopératif dans les systèmes d'interrogation y est brièvement décrit, de mêmes que les options que nous proposons pour fournir des résultats aux requêtes sans réponse. Enfin, ce chapitre présente le résultat de réflexions menées dans le cadre de notre participation à l'ACI APMD[3]. Ces réflexions portent sur la possibilité d'utiliser un vocabulaire d'interrogation différent du vocabulaire de construction des résumés. Elles ont également une portée dans l'interrogation flexible car les systèmes présentés dans le chapitre 3 ne font pas l'hypothèse d'un unique vocabulaire statique.

Les caractéristiques de ces systèmes sont également à l'origine du travail sur les index. Ces systèmes requièrent en effet un accès aux données pour respecter les préférences qui leur sont indiquées. Le chapitre 5 présente donc une synthèse des travaux sur les index de bases de données. Dans un système de gestion de bases de données (SGBD), les index sont utilisés pour accélérer les opérations sur les données, en permettant un accès relativement rapide aux données à traiter. La première partie du chapitre présente d'abord les méthodes d'indexation traditionnelles ainsi que des notions qui servent à caractériser les index. La deuxième partie, plus volumineuse, traite des index dits « multidimensionnels ». Elle présente une sélection de ces techniques et effectue un recensement des propriétés utilisées pour évaluer les index multidimensionnels. Le chapitre, qui se veut une référence rapide pour les personnes intéressées par les index, se termine par une analyse mettant en rapport ces propriétés et les résumés SAINTETIQ.

Le chapitre 6 étudie la faisabilité de l'utilisation de hiérarchies de résumés pour accéder aux données de tables relationnelles. Cette étude expérimentale est menée dans le SGBD relationnel PostgreSQL. Elle consiste à définir une structure d'index à partir d'une hiérarchie de résumés et à implémenter les opérations classiques d'une méthode d'accès (ou technique d'indexation). L'objectif de ce travail est d'aboutir à des conclusions quant aux éléments de performance d'une

[3] *Action Concertée Incitative « Accès Personnalisé à des Masses de Données »* : projet MD 33 2004-2007

structure d'index basée sur les résumés linguistiques que nous traitons. Ainsi, par rétroaction, la construction des résumés pourrait être redéfinie et adaptée à la fonction d'index. Cet objectif n'est pas complètement atteint, mais le chapitre, qui décrit étape par étape l'intégration dans le SGBD PostgreSQL, montre des premiers résultats encourageants.

Ce document se conclut par un bilan de l'ensemble du travail ici présenté et les perspectives envisagées pour sa suite.

CHAPITRE 1

Les résumés du modèle SAINTETIQ

Introduction

Les volumes de données stockés par les systèmes informatiques sont de plus en plus importants en raison de facteurs variés (archivage, avancées technologiques, disponibilité des supports de stockage, interconnexion des systèmes, etc.). Ces conditions rendent plus difficile la recherche des informations pertinentes pour un besoin spécifique. En effet, le traitement de grands volumes de données requiert des outils adaptés, capables, entre autres fonctionnalités, de « passer à l'échelle ».

Un moyen de gérer le problème du volume des données prises en compte dans un traitement informatique consiste à réduire ce volume. Il existe pour cela de nombreuses techniques de réduction du volume des données, dont les résumés de données font partie. La solution du résumé a pour ambition de synthétiser les données, c'est-à-dire en transmettre l'essentiel, mais sous une forme plus compacte. Cependant, elle présente ses propres difficultés : couvrir toutes les données, minimiser le volume des résumés et les présenter sous une forme facilement intelligible.

Le modèle SAINTETIQ, auquel ce chapitre est consacré, est une méthode de construction de résumés de données structurées composées de couples attribut/valeur. À ce titre, le modèle nourrit les mêmes ambitions que toute approche de résumé et fait face aux mêmes difficultés évoquées ci-dessus. Il se distingue néanmoins par deux particularités : le souci d'intelligibilité des résumés construits, et la génération de résumés qui décrivent les données à des niveaux d'abstraction différents.

Pour atteindre ses objectifs, le processus qui sera décrit utilise des techniques diverses. Des domaines comme l'analyse de données, la fouille de données, l'apprentissage automatique ou encore la théorie des sous-ensembles flous, ont permis d'obtenir les résumés linguistiques de données présentés ci-après en sections 1.2 à 1.6. Au préalable, les techniques de réduction de

données sont présentées en section 1.1 afin d'exposer la problématique générale des résumés linguistiques SAINTETIQ.

1.1 Techniques de réduction de données

Cette section est consacrée à une brève présentation générale des techniques de réduction de données, dont l'objectif est de réduire le volume de données en entrée d'un traitement. Malgré une apparente similarité, il est nécessaire de distinguer ces techniques (de réduction des données) des techniques de compression même si ces dernières sont parfois utilisées par la réduction de données.

La compression semble ne viser qu'une réduction du volume des données, traditionnellement liée à des limites matérielles en capacité de stockage ou en débit de transmission. L'application la plus représentative de la compression de données est la compression de fichiers, utilisant des algorithmes (RLE, LZW, codage de Huffman, etc.) ou utilitaires (PkZip, 7-zip, deflate, compress, etc.) relativement connus. Le résultat de ces algorithmes est dit « sans perte » en ce sens qu'un algorithme inverse peut être appliqué au résultat pour retrouver exactement les données initiales. On parle parfois de *compactage* pour désigner cette compression sans perte. La compression peut être également « avec perte » pour certains types de données (audio, vidéo et photo). Dans ce dernier cas, la décompression de données compressées ne garantit pas de reconstituer à l'identique les données initiales, mais le résultat est quasiment identique pour l'oreille ou l'œil humain.

La motivation des techniques de réduction de données est autre. Ces techniques répondent au besoin d'obtenir rapidement des réponses approximatives dans des contextes où « l'obtention d'une réponse exacte est un processus habituellement long » [3] et où une réponse approximative apporte suffisamment d'informations pour être acceptable.

1.1.1 Quelles méthodes ?

Intuitivement, les techniques de réduction de données se ramènent à un traitement effectué sur des données représentatives de l'ensemble des données : le résultat d'une réduction de données est un résumé (des données initiales), qui adopte parfois une forme différente de celle des données initiales.

Le contexte d'une application peut également faire intervenir des techniques de réduction de données. Par exemple, dans les systèmes distribués en mode pair à pair, la volatilité des

pairs a pour conséquence de rendre indisponible une partie des données du système. Des mécanismes de représentation compacte des données externes à un pair doivent être mis en place si l'intention est de répondre aux requêtes de manière complète vis-à-vis du système. Cependant, le mode pair à pair n'est pas une condition nécessaire à l'existence de ces mécanismes dans les systèmes distribués. Pour les systèmes géographiquement distants ou constitués de grappes importantes, par exemple les moteurs de recherche, une forme plus ou moins sophistiquée d'extrapolation est utilisée pour répondre aux requêtes (d'où la formulation « résultats 1-10 sur un total d'environ x pages » de Google ou « 1-10 sur environ x pour ... » de Yahoo Search). La réduction de données se retrouve également dans les caches Internet à travers le concept des *cache digest* [57, 121]. Un *cache digest* est une synthèse du contenu d'un cache Web. Utilisé au sein d'une hiérarchie de caches, il permet de transmettre le contenu d'un cache à d'autres caches, afin de réduire les connexions aux serveurs source de pages Web.

Le contexte des flux de données est un autre exemple [2]. En effet, un flux de données n'a pas de taille connue à un instant donné. Même sans prendre en considération la capacité de stockage nécessairement finie, il est évident qu'un traitement tenant compte de toutes les données aurait un temps de réponse intolérablement long. Ce temps serait infini si des opérateurs *bloquants* étaient utilisés [2] (un opérateur est dit bloquant lorsqu'il lui est nécessaire de connaître toutes les données auxquelles il s'applique pour délivrer un résultat correct – par exemple, MIN ou SUM).

Si l'on considère que les algorithmes et traitements ont une efficacité optimale, la solution évidente lorsqu'on traite de bases de données consiste à réduire l'impact des facteurs pénalisant le temps de réponse, essentiellement la quantité de données. Cet objectif peut être atteint de diverses manières, en particulier en réduisant le nombre d'instances (réduction verticale) ou le nombre d'attributs pris en compte (réduction horizontale). En réduction verticale (par exemple, l'échantillonnage), les instances sont choisies parmi les données initiales, ou construites après transformation des données dans un autre mode de représentation (par exemple, les histogrammes et les techniques de classification [32, 45, 55] en général). En réduction horizontale, les corrélations entre attributs sont utilisées pour « écarter » certains attributs et ne conserver que les plus importants (c'est le cas des méthodes d'analyse factorielle [29, 38]).

1.1.2 Quels types d'approximation ?

L'approximation obtenue en usant d'une technique de réduction de données peut être quantitative : le nombre d'instances ou d'enregistrements est différent de celui obtenu dans un traitement « normal ». Le résultat de la réduction est basé sur des éléments prototypiques – représen-

tatifs des données réelles – accompagnés de données supplémentaires (fréquence, proportion, écart-type, etc.). C'est le cas des méthodes d'interpolation [128] et de l'échantillonnage, en statistiques [39] ou en traitement du signal [109, 112].

Exemple 1 :

Un sous-ensemble flou S, défini sur un domaine numérique continu U par une fonction d'appartenance f_U, couvre en général plusieurs valeurs du domaine sur lequel il est défini. Lorsque le sous-ensemble flou a une forme trapézoïdale, sa représentation se réduit en pratique à quatre valeurs a, b, c, d du domaine. Les valeurs a et d délimitent le support de S, c'est-à-dire l'ensemble des valeurs qui appartiennent plus ou moins à S. On admet que $f_U(a) = 0$ et $f_U(d) = 0$. Les valeurs b et c délimitent le noyau de S, c'est-à-dire l'ensemble des valeurs qui appartiennent totalement à S. On admet alors que $f_U(b) = 1$ et $f_U(c) = 1$. La connaissance du quadruplet (a, b, c, d) et les valeurs (implicites) de f_U pour chacun de ces points permettent de déterminer le degré d'appartenance de tout élément x du domaine U par interpolation linéaire. Par exemple, pour $x \in [a, b]$, $f_U(x) = \frac{x-a}{b-x}$. De même, il est aisé de déterminer les éléments du domaine associés à une valeur α du degré d'appartenance à S, sans avoir à conserver l'ensemble des valeurs du support de S ni les degrés d'appartenance correspondants. La quantité de données décrivant les éléments du sous-ensemble flou est ainsi réduite à quatre valeurs du domaine.

L'approximation peut également être qualitative ; ce sont les données elles-mêmes qui sont approchées. Par exemple, certaines techniques construisent un modèle des données grâce auquel les valeurs de données ou d'enregistrements peuvent être dérivées. Les méthodes de régression ou de transformation discrète en analyse de signaux (ondelettes [110], transformées de Fourier [44, 54, 106], transformées en cosinus discrètes [58, 106, 114]) aboutissent à ce type d'approximations, parfois caractérisées par une (ou plusieurs) valeur typique de la qualité (coefficient de corrélation linéaire, marge d'erreur, etc.).

Exemple 2 :

Supposons que l'on dispose d'un vecteur de 8 valeurs $V = [10, 12, 19, 20, 18, 16, 11, 09]$ issu, par exemple, de l'échantillonnage d'un signal sonore analogique. Parmi les transformées en ondelettes, la plus simple à illustrer est la transformée de Haar (d'après Alfred Haar, en 1909). Cette transformée utilise deux fonctions, S et D pour établir un modèle des données. Supposons que les définitions suivantes sont choisies pour les fonctions S et D :

$$\begin{aligned} S(x, y) &= \frac{x + y}{2} \\ D(x, y) &= \frac{y - x}{2} \end{aligned}$$

La transformée de Haar est le résultat d'un processus récursif qui remplace une paire de valeurs par le résultat des fonctions S et D, résultat arrondi à l'entier le plus proche de zéro pour les

besoins de cet exemple. A la première itération, on obtient deux vecteurs V_S^1 et V_D^1, dont la taille respective est la moitié de celle de V : $V_S^1 = [11, 19, 17, 10]$ et $V_D^1 = [1, 0, -1, -1]$. L'itération suivante utilise V_S^1 comme vecteur initial et produit $V_S^2 = [15, 13]$ et $V_D^2 = [4, -3]$. Par la suite, on a $V_S^3 = [14]$ et $V_D^3 = [-1]$. Le vecteur initial étant réduit à une valeur, la récursion est arrêtée. La transformée est constituée de ce dernier vecteur (V_S^3) et des vecteurs V_D dans l'ordre inverse (V_D^3, V_D^2, V_D^1), soit $V^* = [14, -1, 4, -3, 1, 0, -1, -1]$. Dans cet exemple, la première valeur dans V^* (14) est représentative de celles dans V, les autres valeurs caractérisent la valeur 14.

Un dernier type d'approximation, illustré par la suite de ce chapitre, est celui des méthodes de résumé par agrégation ou classification. Ici, l'objectif est d'obtenir une vision plus globale des données que l'exposition de valeurs spécifiques. Le détail des données, subsidiaire, peut cependant être disponible. Nous classons dans cette catégorie les index de bases de données (qui peuvent être étendus de manière à conserver des informations agrégatives sur les données [1]), les méthodes de classification et les méthodes statistiques de calcul d'agrégats (OLAP [42], Quotient Cube [95]).

Notons qu'une technique de réduction de données peut néanmoins fournir des résultats exacts et non approximatifs. C'est le cas des index, discutés au chapitre 5 pour leur usage principal d'accès aux données. De même, certaines transformées sont réversibles [71, 154] ; le taux d'erreur de l'approximation est alors nul. C'est ainsi que la transformée de Haar dans l'exemple 2 peut permettre de retrouver à l'identique le vecteur initial : il suffirait de choisir des fonctions S et D sans arrondi. Autre exemple, la représentation d'un sous-ensemble flou trapézoïdal par quatre valeurs de son domaine de définition (exemple 1) fournit exactement les mêmes informations qu'une énumération complète de ses éléments.

1.1.3 Résumés de données

Les méthodes de résumé de données structurées réalisent également une approximation des données auxquelles elles s'appliquent, c'est-à-dire des enregistrements de bases de données. Ces méthodes ont comme point commun de recourir à la théorie des sous-ensembles flous, proposée par Zadeh [151], pour exprimer des concepts vagues. Pour ce faire, des termes du langage naturel sont modélisés par des sous-ensembles flous, ce qui permet de décrire les valeurs d'attribut par ces termes. Les descriptions obtenues matérialisent les concepts qui existent au sein des enregistrements, par exemple, « les personnes jeunes et bien payées ». Chacun des enregistrements est rattaché à un ou plusieurs concepts suivant l'adéquation entre ses valeurs d'attribut et les termes qui étiquettent le concept (*jeune* et *bien payé*). Ces méthodes s'inscrivent dans le cadre des résumés linguistiques de Yager [150].

On trouve parmi ces méthodes les résumés quantifiés et les résumés à base de règles floues. Les résumés quantifiés [111, 117] utilisent des quantificateurs flous pour décrire les données. Par exemple, dans SUMMARYSQL [117], l'évaluation du résumé « *summary* la plupart *from* Employés *where* âge est jeune » fournit un degré de validité de la proposition « la plupart des employés sont jeunes ». Ce degré caractérise la mesure dans laquelle la proposition est satisfaite par les données. Les résumés à base de règles floues sont découverts par recherche d'associations entre attributs ([26]) ou en exploitant des dépendances fonctionnelles floues [21, 46]. Ils permettent d'obtenir, dans le cas des règles graduelles de Bosc *et al.* [26], des propositions sous la forme « plus l'âge des employés est *âgé*, plus leur salaire est *élevé* ».

Il est également possible de résumer des enregistrements par généralisation successive des descriptions. La structure de données obtenue est alors une hiérarchie. C'est le cas des hiérarchies « is-a » de Lee et Kim [97] et des résumés du modèle SAINTETIQ [115]. Les résumés de SAINTETIQ se distinguent en utilisant un même ensemble de termes linguistiques (le *vocabulaire*) sans assignation de niveau. Les descriptions générées sont donc unifiées sur le plan des termes (ou *étiquettes*) linguistiques. Ce modèle procède à un lissage des données grâce à une abstraction par le vocabulaire, puis à une classification. Ces deux opérations sont décrites dans la suite de ce chapitre. On obtient ainsi des descriptions à plusieurs niveaux de détail différents. Dans ce sens, les résumés réalisent une approximation du dernier type évoqué ci-dessus. La procédure de recherche permettant d'atteindre ce résultat fait l'objet du chapitre 2. Mais il est également possible d'utiliser les résumés comme index pour accéder aux enregistrements d'une base de données et fournir des résultats exacts ; ceci est réalisé au chapitre 6.

1.2 Données du processus de résumé SAINTETIQ

Les informations en entrée du processus de résumé sont de deux types : les données à résumer, et celles, désignées par « connaissances de domaine », donnant des indications sur la manière de les résumer.

Les données à résumer sont des données relationnelles au sens des bases de données relationnelles. À ce titre, elles sont organisées en enregistrements (ou « tuples ») qui suivent le schéma d'une relation R définie sur un ensemble d'attributs $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$. Chaque attribut $\mathcal{A}_i$ est défini sur un domaine, noté $D_{\mathcal{A}_i}$, qui peut être numérique ou symbolique. Ainsi, un enregistrement t est un tuple formé d'une suite de valeurs suivant l'ordre prédéfini des attributs A_i. Il est noté :

$$\langle t.A_1, t.A_2, \ldots, t.A_n \rangle \ .$$

Par exemple, pour une relation R = (épaisseur, dureté, température), un enregistrement $t = \langle$ 10, 38, 900 $\rangle$ présente une valeur du premier attribut ($t.\texttt{épaisseur} = 10$), puis une valeur du deuxième ($t.\texttt{dureté} = 38$) et une valeur du troisième ($t.\texttt{température} = 900$).

Une autre contrainte sur les données est leur complétude : toutes les valeurs d'attributs doivent être présentes. Pour tout enregistrement t d'une relation R, la valeur $t.A_i$ est nécessairement connue, élémentaire, précise et certaine. Les données incomplètes, incertaines ou mal connues ne sont donc pas traitées. En outre, les éventuelles relations entre enregistrements (dépendances fonctionnelles, liens hiérarchiques, etc.) que pourraient gérer des bases de données objet par exemple, ne sont pas prises en compte.

Les *connaissances de domaine* régissent l'interprétation qui sera faite des valeurs d'attributs dans la constitution des résumés. Elles sont constituées essentiellement de variables linguistiques définies sur les domaines d'attributs de la relation résumée. Elles sont données par l'utilisateur ou un expert, ceci afin de définir un langage de description des données dont la sémantique soit la plus proche possible de l'utilisateur. Ainsi, les connaissances de domaine fournissent le vocabulaire d'expression des résumés.

Les variables linguistiques, introduites par Zadeh en 1975 [152], permettent ici de décrire les valeurs d'un domaine d'attribut grâce à des caractérisations floues. Si A est un attribut et D_A son domaine, on écrit habituellement « $t.A = x$ » avec x une valeur du domaine D_A. En utilisant une variable linguistique, la valeur $t.A$ du tuple t sur l'attribut A n'est plus une valeur spécifique x : on écrira « $t.A = d$ » ou « $t.A$ est d » avec d un descripteur linguistique (par exemple, « mince » dans la figure 1.1) issu de la variable linguistique sur l'attribut A.

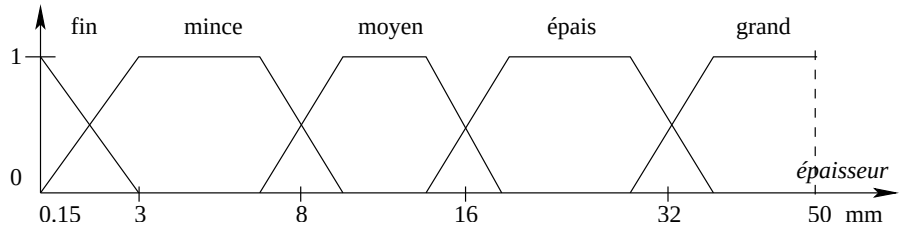


Figure 1.1 – Variable linguistique définie sur le domaine de l'attribut épaisseur

Formellement, une variable linguistique est représentée par un triplet (A, U, $\mathcal{D}$) avec U le domaine de l'attribut A et $\mathcal{D} = \{d_i, 1 \leq i \leq n\}$, un ensemble de sous-ensembles flous de U [28]. Chaque terme d_i est muni d'une fonction d'appartenance f_{d_i} définie sur U et à valeurs dans l'intervalle $[0, 1]$. Un sous-ensemble flou d_i peut également être une partie ordinaire du domaine U [28], c'est-à-dire un ensemble classique (sa fonction d'appartenance n'est plus

graduelle mais binaire). On dit que $\mathcal{D}$ définit une partition de U lorsque les d_i vérifient les propriétés :

- de couverture : $\forall\, x \in U, \exists\, i / f_{d_i}(x) > 0$ et
- de contraste : $\forall\, x \in U, f_{d_p}(x) = 1 \wedge f_{d_q}(x) = 1 \Rightarrow d_p = d_q$).

La partition $\mathcal{D}$ est dite « forte » si on a en plus $\forall\, x \in U, \sum_{i=1}^{n} f_{d_i}(x) = 1$.

Sur la figure 1.1, l'attribut `épaisseur` instancie A sur le domaine $U =$ [0.15 mm, 50 mm], avec l'ensemble des descripteurs $\mathcal{D} = \{$*fin*, *mince*, *moyen*, *épais*, *grand*$\}$. Les éléments de $\mathcal{D}$ sont appelés des *termes*, des *descripteurs* ou des *étiquettes linguistiques*.

Lorsque le domaine D_A est discret, les valeurs discrètes de D_A sont munies de leur degré d'appartenance à l'étiquette floue du vocabulaire de description des données. Le tableau 1.1, tiré de [125], donne un exemple des sous-ensembles flous pour un attribut discret (`Activité`). Les termes du vocabulaire de description sont *artiste*, *homme d'affaires*, *cadre supérieur*, *employé*, *commerçant*, *chef d'entreprise* et *sans activité*. Les valeurs discrètes du domaine des données sont celles de la colonne de droite (*saxophoniste*, *barman*, *femme au foyer*, etc.).

Table 1.1 – Sous-ensembles flous sur l'attribut `Activité`

Étiquette floue	Fonction d'appartenance
artiste	$\{1.0$/*saxophoniste* + 0.7/*baby star* + 0.3/*présentateur* + 0.9/*clown* + 0.9/*danseuse exotique* + 0.4/*cambrioleur*$\}$
homme d'affaires	$\{0.7$/*commerçant* + 0.7/*barman* + $0.6/procureur$ + 0.9/*patron de centrale nucléaire* + 0.8/*assistant de direction* + 0.8/*présentateur* + 0.9/*cambrioleur*$\}$
cadre supérieur	$\{1.0$/*avocat* + 0.8/*inspecteur de la sécurité* + 0.8/*assistant de direction*$\}$
employé	$\{1.0$/*secrétaire* + 0.3/*danseuse exotique*$\}$
commerçant	$\{1.0$/*commerçant* + 0.9/*barman*$\}$
chef d'entreprise	$\{1.0$/*patron de centrale nucléaire* + 0.2/*femme au foyer*$\}$
sans activité	$\{1.0$/*sans emploi* + 0.7/*retraité* + 0.5/*femme au foyer* + 0.4/*baby star* + 0.4/*cambrioleur*$\}$

D'une manière générale, les connaissances de domaine permettent de faire la correspondance entre les valeurs de domaines d'attribut et un vocabulaire d'expression des résumés de données. Un domaine D_A est alors réécrit par l'ensemble, noté D_A^+, des termes qui lui sont applicables. Par exemple, $D^+_{\texttt{Epaisseur}} =$ {fin, mince, moyen, épais, grand} et $D^+_{\texttt{Activité}} =$ {artiste, homme d'affaires, cadre supérieur, employé, commerçant, chef d'entreprise, sans activité}.

1.3 Construction des résumés

La construction des résumés peut être vue comme un processus de découverte de connaissance. En entrée, les enregistrements de la relation R à résumer sont associés aux connaissances de domaine. En sortie, les résumés obtenus décrivent les données par le biais du vocabulaire, c'est-à-dire l'ensemble des termes, fourni par les connaissances de domaine. Le processus est incrémental : les tuples sont pris en compte l'un après l'autre. Il peut donc être réduit au traitement d'un tuple t. Une hiérarchie de résumés est obtenue après répétition de ce traitement pour chacun des enregistrements de R.

L'architecture de construction des résumés proposée dans [125] est une implémentation orientée service du modèle SAINTETIQ pour le résumé en ligne de données relationnelles. Cette architecture distingue deux services au sein du processus : un service de réécriture et un service de résumé.

1.3.1 Service de réécriture

Ce service réalise une abstraction des données grâce aux connaissances de domaine. On obtient une représentation des données sous forme de sous-ensembles flous (voir section 1.2). Cette nouvelle représentation est homogène quelle que soit la nature du domaine d'attribut. Le principe de l'abstraction est de négliger les différences non significatives entre valeurs d'attributs. Ainsi, les valeurs 45 mm et 48 mm, désignant l'épaisseur d'une plaque de métal, bien que distinctes, seront décrites par la même étiquette « grand » (voir figure 1.1).

Il s'agit, en réécrivant les données brutes, de trouver toutes les réécritures possibles d'un tuple t d'après les connaissances de domaine. L'objectif est d'offrir un traitement doté d'une propriété de complétude et de rester fidèle au contexte sémantique du processus de résumé.

Soient le tuple $t = \langle t.A_1, t.A_2, \ldots, t.A_n \rangle$ et Φ_A la fonction générique associant toute valeur d'attribut $t.A$ aux descripteurs pouvant caractériser la valeur $t.A$. L'opération de réécriture construit un tuple :

$$t' = \langle \Phi_{A_1}(t.A_1), \Phi_{A_2}(t.A_2), \ldots, \Phi_{A_n}(t.A_n) \rangle \ .$$

Notons que t' peut être multivalué. Dans ce cas, le modèle effectue une réduction de t' à des attributs monovalués. Ainsi, la réécriture d'un unique tuple peut engendrer plusieurs représentations distinctes désignées par le terme « tuple candidat » ; on note $\varphi(t)$ l'ensemble des tuples candidats (monovalués) engendrés par la réécriture du tuple t. Le tableau 1.2 présente des tuples candidats d'enregistrements de la relation $\mathsf{R} = (\texttt{épaisseur}, \texttt{dureté}, \texttt{température})$.

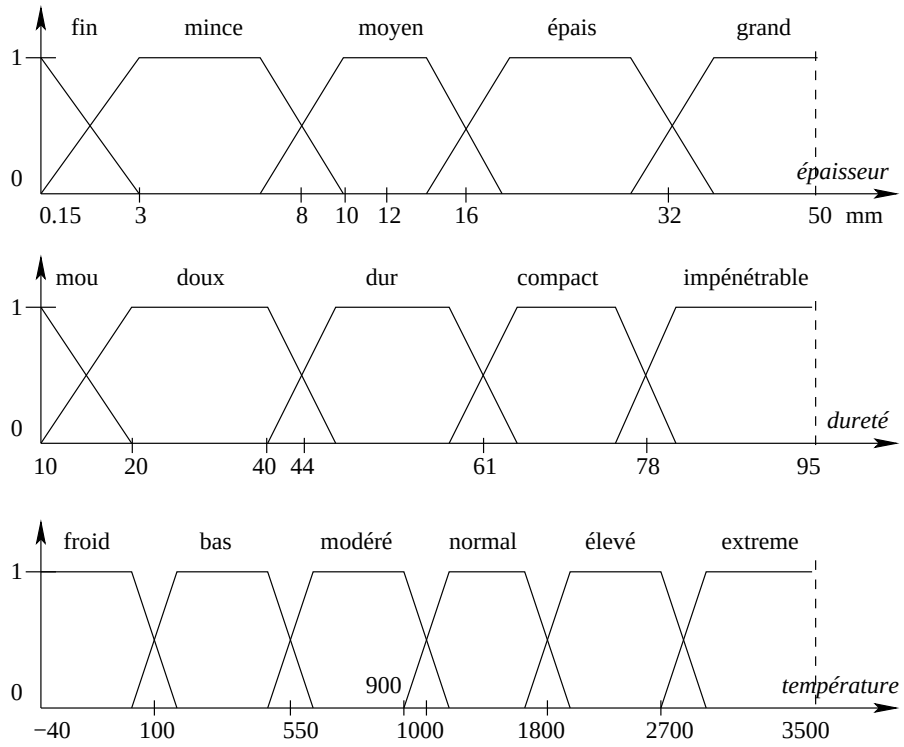


Figure 1.2 – Variables linguistiques de $\mathsf{R} = (\texttt{épaisseur}, \texttt{dureté}, \texttt{température})$

Table 1.2 – Exemple de réécriture

Alliage	N-uplet	Traduction
UZ40 (laiton)	$t_l = \langle 10, 38, 900 \rangle$	$\varphi(t_l) = \{t_{l1}\}$ avec $t_{l1} = \langle 1.0/\text{moyen}, 1.0/\text{doux}, 1.0/\text{modéré} \rangle$
CuSn12 (bronze)	$t_b = \langle 8, 40, 850 \rangle$	$\varphi(t_b) = \{t_{b1}, t_{b2}\}$ avec $t_{b1} = \langle 0.5/\text{moyen}, 1.0/\text{doux}, 1.0/\text{modéré} \rangle$, $t_{b2} = \langle 0.5/\text{mince}, 1.0/\text{doux}, 1.0/\text{modéré} \rangle$
CuAs05 (cuivre dopé)	$t_c = \langle 12, 44, 896 \rangle$	$\varphi(t_c) = \{t_{c1}, t_{c2}\}$ avec $t_{c1} = \langle 1.0/\text{moyen}, 0.5/\text{doux}, 1.0/\text{modéré} \rangle$, $t_{c2} = \langle 1.0/\text{moyen}, 0.5/\text{dur}, 1.0/\text{modéré} \rangle$

1.3.2 Service de résumé

Le service de résumé réalise la classification conceptuelle des données en résumés et la structuration des résumés en hiérarchie. Une relation d'ordre partiel est à la base de l'organisation en arbre. Elle est explicitée en section 1.5.2.

Il s'agit, pour le service de résumé, de prendre en compte les tuples candidats obtenus en sortie du service de réécriture, c'est-à-dire, de les incorporer dans l'arbre des résumés. Un tuple candidat ct suit un chemin partant de la racine z_0 pour atteindre un résumé z_f dont la description est identique à celle de ct. Cependant, le parcours de ct dans l'arbre est découvert au fil de sa progression grâce à des opérateurs d'apprentissage. Le rôle de ces opérateurs est de modeler la hiérarchie de résumés en fonction des données déjà incorporées et donc, de la forme courante de la hiérarchie.

À chaque nœud z traversé par ct, le service de résumé décide de l'opérateur dont l'application fournit les meilleurs résultats suivant des mesures de dispersion et de spécificité qui servent de critères d'évaluation. Par cette opération, le prochain nœud sur le chemin menant à la feuille z_f est déterminé : c'est celui pour lequel la dispersion des données (après l'incorporation de ct) sera minimale. De même, la mesure de spécificité détermine si z est suffisamment spécifique pour prétendre au statut de feuille. Si tel est le cas, l'incorporation de ct est terminée. Autrement, l'opération recommence avec le nouveau nœud.

Les quatre opérateurs d'apprentissage du modèle proposé par Raschia [115] sont :

1. *initialiser* : un nouveau résumé z_* est créé et ne contient pour l'instant que ct. z_* est nécessairement une feuille.

2. *affecter* : cet opérateur matérialise le fait qu'un résumé z existant est traversé par ct. La description de z peut être modifiée pour rendre compte de l'existence des caractéristiques de ct.

3. *fusionner* : deux résumés z_1 et z_2 sont jugés trop spécifiques au regard de leur position (trop élevée) dans l'arbre. La fusion (voir figure 1.3) donne naissance à z_{merge}, nouveau parent de z_1 et z_2, qui descendent ainsi d'un niveau dans l'arbre.

4. *scinder* : à l'inverse de l'opérateur précédent, celui-ci permet de supprimer un résumé z trop général par rapport à sa position dans l'arbre (voir figure 1.4 pour une illustration de la scission du résumé z_1). Il n'est pas question de remonter z au niveau supérieur de la hiérarchie car le parent de z couvre déjà les données de z et présente donc les caractéristiques qui décrivent ces données ; le résumé z, devenu inutile, doit être supprimé.

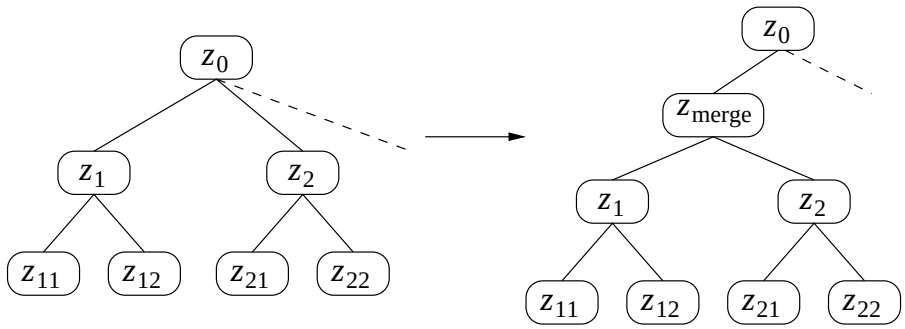


Figure 1.3 – Opérateur de fusion

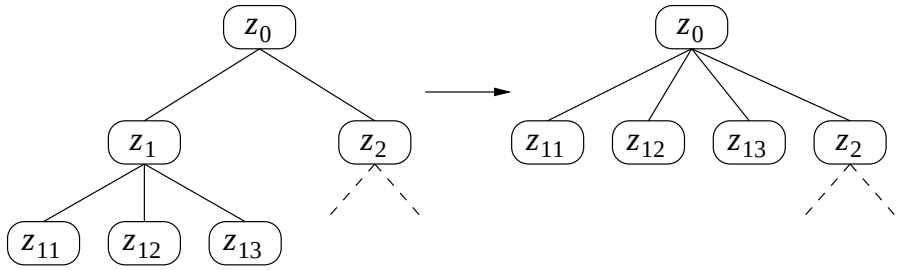


Figure 1.4 – Opérateur de scission

1.4 Expression des résumés

1.4.1 Extension et intension

Les représentations adoptées pour les résumés décrivent les enregistrements couverts à la manière des définitions d'ensembles mathématiques : en extension et en compréhension. La description en extension est, en général, sans grand intérêt pour un utilisateur. C'est en effet la liste des enregistrements qui participent au résumé. Suivant le jeu de données, certains résumés peuvent présenter une liste suffisamment courte pour qu'une manipulation par l'utilisateur soit possible. Cela peut être le cas pour les niveaux inférieurs de l'arbre, notamment les feuilles. Néanmoins, la représentation en extension est plus adaptée à un traitement informatique par un programme, plus apte à gérer de longues listes. Elle n'a pas d'écriture formalisée, mais peut être notée comme un ensemble des enregistrements qu'il représente. La représentation en extension d'un résumé z est :

$$\mathsf{R}_z = \{ct_1, ct_2, ct_3, ct_4\} \ .$$

La représentation en intension d'un résumé correspond à une définition en compréhension : elle fait apparaître les propriétés permettant de déterminer l'appartenance d'un élément à l'ensemble. Par exemple, l'ensemble des entiers naturels $\mathbb{N}$ peut être défini (par rapport à l'ensemble $\mathbb{Z}$ des entiers relatifs) par $\mathbb{N} = \{n \in \mathbb{Z} / n \geq 0\}$. Dans le cas des résumés, les caractéristiques des données sont décrites par des étiquettes linguistiques. Une représentation en intension d'un résumé z s'écrit sous la forme :

$$z = \langle \{\alpha_1^1/d_1 + \alpha_2^1/d_2 + \ldots + \alpha_n^1/d_n\}, \{\ldots\}, \{\alpha_1^p/d_1 + \alpha_2^p/d_2 + \ldots + \alpha_m^p/d_m\} \rangle \ .$$

Dans cette formalisation, p est le nombre d'attributs de la relation résumée. Pour chaque attribut, une expression sous la forme $\alpha_1^1/d_1 + \alpha_2^1/d_2 + \ldots + \alpha_n^1/d_n$ fait partie de l'intension. Chaque d_i représente un descripteur issu de la variable linguistique décrivant l'attribut A_i. Les indices n et m révèlent le fait que les attributs de résumés sont en général multivalués. Cependant, les feuilles d'une hiérarchie de résumés ne présentent qu'un seul descripteur par attribut.

Exemple 3 :
Soit la relation R = (`épaisseur`, `dureté`, `température`). *Un résumé feuille de la hiérarchie construite sur* R *est monovalué sur tous les attributs. Par exemple, le résumé* z_{11} *de la figure 1.3 est :*

$$z_{11} = \langle 1.0/\textit{mince}, 0.8/\textit{mou}, 1.0/\textit{normale} \rangle \ .$$

En revanche, le résumé z_1*, père de* z_{11}*, est nécessairement multivalué sur au moins un attribut (par exemple, l'épaisseur) car il est obtenu par l'union de* z_{11} *et* z_{12}*. Un résumé est dit « multivalué » s'il présente, dans son intension, un ou plusieurs attributs ayant plus d'un descripteur.* z_1 *est multivalué car l'attribut* `épaisseur` *est caractérisé par deux descripteurs (*mince *et* fin*) :*

$$z_1 = \langle 1.0/\textit{mince} + 0.5/\textit{fin}, 0.8/\textit{mou}, 1.0/\textit{normale} \rangle \ .$$

La notation « $1.0/\text{mince}+0.5/\text{fin}$ » rend compte du fait qu'un nœud généralise ses nœuds fils et décrit les mêmes données que ces derniers. L'ensemble de ses descripteurs est donc l'union des ensembles de descripteurs de ses fils.

On notera que l'expression d'un résumé en intension fait apparaître, outre les étiquettes linguistiques, des valeurs réelles attachées à chaque étiquette (« 1.0/*mince* » ou « 0.5/*fin* » dans l'exemple 3 ci-dessus). Ces valeurs dans une intension sont des degrés de satisfaction (voir la section 1.4.2.1 ci-après). Notons cependant que toutes sortes de valeurs, statistiques notamment, peuvent être attachées aux résumés suivant le besoin. Les degrés et mesures proposés dans le modèle SAINTETIQ sont détaillés ci-dessous.

1.4.2 Degrés et mesures

1.4.2.1 Degrés de satisfaction

Un degré de satisfaction d'un descripteur, comme un degré d'appartenance d'un élément à son ensemble (voir section 1.2), prend ses valeurs dans l'intervalle réel $[0, 1]$. Un degré d'appartenance caractérise l'élément dans le référentiel de l'ensemble ; un degré de satisfaction ω d'une étiquette linguistique d indique à quel point d représente l'élément. L'élément appartient à d avec un degré d'appartenance $\alpha = \omega$. Le degré de satisfaction équivaut au maximum des degrés d'appartenance des valeurs d'attribut au sous-ensemble flou du descripteur, étant donné que :

- un résumé couvre a priori une multitude de tuples ;
- plusieurs tuples peuvent être décrits par la même étiquette, avec des degrés d'appartenance différents ;
- l'intension du résumé ne présente qu'un unique degré de satisfaction.

Ainsi, l'expression « $1.0/$normale » dans $z_1 = \langle 1.0/\text{mince}+0.5/\text{fin}, 0.8/\text{mou}, 1.0/\text{normale}\rangle$ indique que l'un au moins des enregistrements dans l'extension de z_1 présente une valeur de température comprise dans l'intervalle $[1.200, 1.600]$ (voir figure 1.5). *normale* est un descripteur « totalement satisfait » par l'ensemble d'enregistrements qu'il représente.

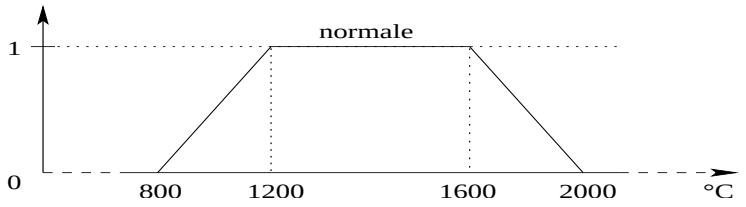


Figure 1.5 – Descripteur *normale* sur l'attribut *Température*

1.4.2.2 Représentativité

La mesure de représentativité d'un résumé z, notée $\mathbf{card}(z)$, caractérise un résumé par rapport à la relation R. Elle rend compte de la proportion de la relation R (en nombre de tuples candidats) couverte par le résumé. Cette mesure statistique fait en effet appel au poids d'un tuple candidat ct défini par :

$$w(ct) = \frac{1}{|\varphi(t)|} \ ,$$

où $w(ct)$ est la proportion du tuple t de R contenue dans ct, élément de la réécriture $\varphi(t)$ de t.

Quant à la représentativité $\mathbf{card}(z)$, elle indique la somme des poids $w(ct)$ des candidats incorporés dans z :

$$\mathbf{card}(z) = \sum_{ct \in \mathsf{R}_z} w(ct) \ .$$

Exemple 4 :
Soit un tuple $t_1 = \langle 24, 70, 1100 \rangle$ *de la relation* $\mathsf{R} = (\texttt{épaisseur}, \texttt{dureté}, \texttt{température})$.
D'après les variables linguistiques de la figure 1.2, t_1 *est réécrit par deux tuples candidats* $ct_1 = \langle 1.0/\textit{épais}, 1.0/\textit{compact}, 0.3/\textit{modéré} \rangle$ *et* $ct_2 = \langle 1.0/\textit{épais}, 1.0/\textit{compact}, 0.7/\textit{normal} \rangle$.
On a les représentativités $w(ct_1) = 0.5$ *et* $w(ct_2) = 0.5$.
Considérons un résumé z_0 *dont l'extension est* $\mathsf{R}_{z_0} = \{ct_1, ct2\}$. *La cardinalité* $\mathbf{card}(z_0)$ *vaut* 1 *car* $\mathbf{card}(z_0) = w(ct_1) + w(ct_2)$.

1.4.2.3 Cardinalité normalisée

La cardinalité normalisée est une mesure de proportion liée à un descripteur particulier d dans le contexte d'un résumé z. Sa portée est limitée à l'attribut correspondant à d. Cette limitation est nécessaire car le même descripteur peut être utilisé dans plusieurs variables linguistiques. C'est le cas de descripteurs courants tels *moyen* ou *normal*. Il serait incohérent d'associer plusieurs étiquettes dans une même mesure au seul motif d'une même représentation textuelle alors qu'elles sont sémantiquement différentes.

La cardinalité normalisée indique la proportion des éléments de z décrits par d sur l'attribut A. Cette mesure ne tient pas compte de la pertinence de la description, c'est-à-dire du degré de satisfaction lié au descripteur d. Elle est notée $\overline{\mathbf{card}}_{z.A}(d)$ et s'exprime par :

$$\overline{\mathbf{card}}_{z.A}(d) = \frac{\mathbf{card}_{z.A}(d)}{|\mathsf{R}_z|} \ ,$$

où $\mathbf{card}_{z.A}(d)$ est le nombre de tuples candidats de z décrits par d sur A et R_z est l'extension de z.

Exemple 5 :
Reprenons l'exemple 4 ci-dessus. La cardinalité normalisée du descripteur « modéré » dans z_0 *est :*

$$\mathbf{card}_{z_0.\texttt{temp}}(\textit{modéré}) = \frac{1}{2} \ .$$

1.5 Propriétés des résumés

Dans cette section, nous présentons les propriétés générales les plus importantes des résumés. Ces propriétés constituent les fondements de notre travail sur l'interrogation. D'autres propriétés, spécifiques au contexte des index multidimensionnels, seront évoquées plus avant (chapitre 5, section 5.2.3).

1.5.1 Unicité d'un résumé

Le processus de construction d'une hiérarchie de résumés est stable : dans le même contexte (connaissances de domaine, schéma de relation R et tuples de R), la hiérarchie obtenue, réduite aux feuilles, est strictement la même. Un tuple candidat ct dérivé d'un tuple t de R par réécriture (section 1.3.1), fait partie d'un unique regroupement. Il n'y a donc qu'une feuille z_f prenant ct en compte dans son extension. Dès son incorporation dans la racine, ct est destiné à rejoindre son regroupement le plus fin, c'est-à-dire z_f. Une hiérarchie de résumés étant un arbre et non un graphe, il n'y a pour ct qu'un *chemin d'incorporation* menant à z_f.

Par conséquent, chaque regroupement élémentaire, représenté par une feuille dans l'arbre, est différent des autres. Au niveau supérieur, les résumés (moins spécifiques) prennent en compte des feuilles différentes. Il s'ensuit que les résumés de niveau supérieur sont eux-mêmes tous différents. À tous les niveaux, les résumés sont différents les uns des autres.

De même, deux résumés quelconques, c'est-à-dire sans contrainte portant sur leur niveau dans l'arbre, sont différents. En effet, pour qu'ils ne le soient pas, il aurait fallu qu'ils partagent les mêmes tuples candidats. Ce qui ne peut arriver car les tuples candidats se retrouvent au niveau d'une feuille et ne font partie des résumés de niveau supérieur que par la transitivité de la relation d'appartenance.

La relation suivante, où $\mathcal{Z}$ désigne l'ensemble des résumés de la hiérarchie, est donc vérifiée :

$$\forall\, z, z' \in \mathcal{Z}, \quad z \neq z' \Leftrightarrow \mathsf{R}_z \neq \mathsf{R}_{z'} \ .$$

1.5.2 Relation d'ordre partiel

Nous avons mentionné, en section 1.3.2, que les résumés d'une hiérarchie sont liés par une relation d'ordre partiel, à savoir leur spécificité ou leur caractère général, ces deux notions étant l'inverse l'une de l'autre. La spécificité d'un résumé augmente avec sa profondeur dans l'arbre. Ainsi, les feuilles sont les résumés les plus spécifiques et la racine est le résumé le

moins spécifique. La figure 1.6 illustre que le résumé z_1 est plus spécifique que z_0 mais moins que z_{11} et z_{12}. L'ordre partiel ne permet de comparer que des éléments d'un même chemin. Par exemple, z_{11} et z_{12} sont incomparables.

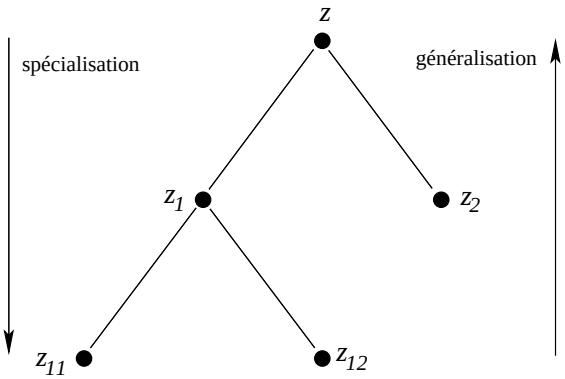


Figure 1.6 – Ordres partiels sur les résumés

La spécificité dénote le fait que, par rapport à son résumé parent z_0, le résumé z_1 est plus précis dans la description de ses éléments. Elle peut également être perçue comme le pouvoir de discrimination d'un résumé vis-à-vis de la relation résumée. Il en découle qu'un résumé z_1 plus spécifique que z_0 est moins *peuplé* que ce dernier. La représentativité de z_1 est inférieure à celle de z_0 : $\mathbf{card}(z_1) < \mathbf{card}(z_0)$.

Une conséquence de l'ordre partiel porte sur les extensions de résumés comparables. Lorsque z_0 est le parent de z_1, les données prises en compte dans z_1 le sont également dans z_0, l'inverse étant toujours faux. Par suite, l'extension de z_1 est incluse dans celle de z_0 : $\mathsf{R}_{z_1} \subset \mathsf{R}_{z_0}$. L'extension de z_1 est donc moins importante en nombre de tuples candidats : $|\mathsf{R}_{z_1}| < |\mathsf{R}_{z_0}|$.

Sur le plan des expressions en intension, l'ordre partiel se traduit par le fait que les descripteurs d'un résumé se retrouvent dans l'intension de tous ses ascendants. À l'inverse, l'intension d'un résumé peut être reconstituée dès que toutes les feuilles dans son sous-arbre sont connues. Ainsi, une étiquette d fait partie de l'intension d'un résumé z seulement si elle fait partie de l'intension d'une feuille de z. Réciproquement, une étiquette d d'un résumé z indique qu'il existe une feuille descendant de z qui couvre des données décrites par d.

La relation de spécificité peut être considérée comme une *spécialisation* du nœud parent (ou, à l'inverse, comme une généralisation des nœuds fils). Par rapport à un nœud frère, la spécialisation du parent se fait sur des sous-ensembles d'attributs différents : z_1 et z_2 présentent sur au moins un attribut, des ensembles flous non-égaux :

$$\forall\, z, z_1, z_2 \;/\; (z \preceq z_1 \wedge z \preceq z_2 \wedge z_1 \not\equiv z_2),\; \exists\, i,\; 1 \leq i \leq n \;/\; z_1.A_i \neq z_2.A_i \qquad (1.1)$$

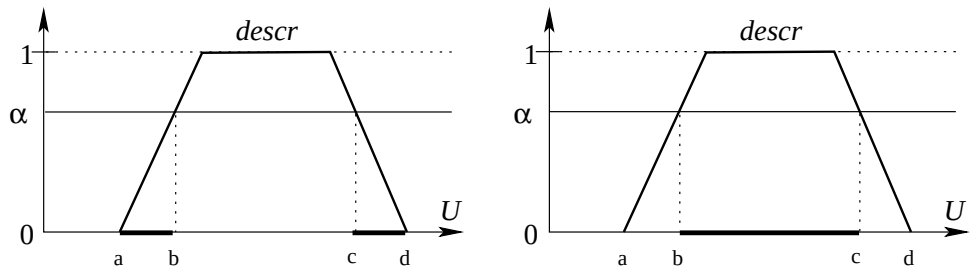
où « $\preceq$ » représente la relation d'ordre partiel avec une sémantique de généralisation et « $\not\equiv$ », le caractère non-comparable de deux résumés.

La section 1.3.2 évoque le fait que l'évolution de l'arbre des résumés dépend, en plus des opérateurs d'apprentissage, d'une mesure de dispersion et d'une mesure de spécificité. Ces mesures sont instanciées de telle sorte que la généralisation d'une intension se fait par ajout de descripteurs ; c'est le sens de l'expression (1.1). Considérons par exemple le résumé $z_0 = \langle\{1.0/\text{mince}\}, \{0.8/\text{mou}\}, \{1.0/\text{normale}\}\rangle$. Il peut être généralisé par $z_1 = \langle\{1.0/\text{mince} + 0.5/\text{fin}\}, \{0.8/\text{mou}\}, \{1.0/\text{normale}\}\rangle$ ou $z_2 = \langle\{1.0/\text{mince}\}, \{0.8/\text{mou}\}, \{1.0/\text{normale} + 0.5/\text{modérée}\}\rangle$ car de nouvelles étiquettes apparaissent dans z_1 et z_2. z_1 et z_2 sont donc des parents possibles pour z_0. Mais $z_3 = \langle\{1.0/\text{mince}\}, \{1.0/\text{mou}\}, \{1.0/\text{normale}\}\rangle$ n'est pas différent de z_0, et ce malgré le degré de satisfaction différent sur l'étiquette *mou*. z_3 n'est en effet, d'après la propriété d'unicité, qu'une nouvelle version de z_0 après la prise en compte d'un tuple candidat générant le degré 1.0 sur le descripteur *mou*.

1.6 Sémantique des résumés

Un résumé est, dans sa représentation intensionnelle, une description des données qu'il couvre. La description est exprimée en utilisant des termes fournis par l'utilisateur, par exemple sur l'attribut A, $z.A = \{\alpha_1/d_1 + \alpha_2/d_2\}$. Le degré de satisfaction α_i qui accompagne chaque étiquette linguistique révèle qu'il existe, au sein des données, un enregistrement dont la valeur sur A est décrite par l'étiquette au degré α_i. En outre, α_i indique les plages de valeurs que peut prendre la valeur $t.A$ d'un tuple sur l'attribut A. La figure 1.7 montre les plages de valeurs délimitées par un degré α suivant que le degré désigne le maximum des degrés de satisfaction ou le minimum. Les plages délimitées sont $]a, b] \cup [c, d[$ (fig. 1.7-a) pour le maximum (c'est l'interprétation adoptée dans le modèle SAINTETIQ) et $[b, c]$ pour le minimum (fig. 1.7-b).

Sur un autre plan, l'intension est exprimée sur chaque attribut de manière indépendante : aucune corrélation n'existe entre deux descripteurs. Ainsi, l'expression $z_4 = \langle\{1.0/\text{mince} + 0.5/\text{fin}\}, \{0.8/\text{mou}\}, \{1.0/\text{normale} + 0.5/\text{modéré}\}\rangle$ n'apporte pas d'information certaine sur la description d'un même enregistrement par deux étiquettes d'attributs différents. Par exemple, on déduit de z_4 que certains tuples sont bien décrits (avec un degré 1) par l'étiquette *mince* sur l'épaisseur. De même, certains tuples sont décrits par l'étiquette *modéré* pour ce qui est de la température. Mais on ne peut encore en déduire que certains tuples sont décrits à la fois par *mince* et par *modéré*. Ce problème d'existence de combinaisons de descripteurs est dû à une ambigüité : des valeurs issues de plages différentes conduisent à une même représentation en



a) au maximum α b) au minimum α

Figure 1.7 – Plages de valeurs induites par un degré de satisfaction α sur un domaine U

intension. Cette ambigüité est mise en évidence dans la figure 1.8 qui montre les espaces de valeurs (zones hachurées) des tuples initiaux pour un résumé $z = \langle\{\alpha/d_1\}, \{\beta/d_2\}\rangle$ défini sur deux domaines d'attributs U et V. Tout ensemble constitué des zones hachurées I, II, III ou IV conduit à l'intension de z. Par exemple, {I}, {II}, {II, IV} ou encore {I, III, IV}.

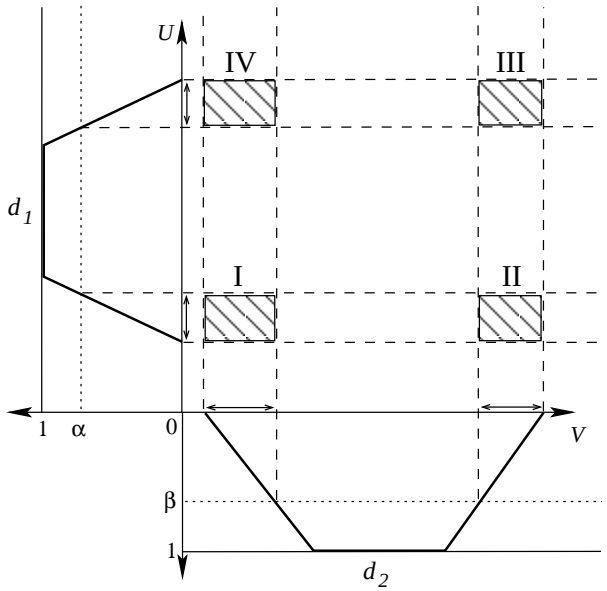


Figure 1.8 – Espaces de valeurs induits par l'intension $z = \langle\{\alpha/d_1\}, \{\beta/d_2\}\rangle$

Une zone hachurée peut être interprétée comme un concept flou. Un résumé est donc la conjonction des concepts flous partageant la même expression en intension.

1.7 Conclusion

Dans ce chapitre, les résumés linguistiques du modèle SAINTETIQ ont été présentés de manière à couvrir les aspects de leur construction et de leur sémantique utiles à la compréhension de la suite de ce document. La problématique des résumés linguistiques a permis d'exposer les objectifs du modèle : fournir une description concise de données relationnelles dans un formalisme facilement intelligible. Puis, les types de données admis en entrée du processus de résumé ont été décrits, de même que les connaissances de domaine, dont le rôle est de fixer le cadre d'interprétation des résumés. Le déroulement du processus de résumé est abordé dans ses deux phases : la réécriture, qui réalise une abstraction des données vers un formalisme homogène quels que soient les domaines d'attributs considérés, et la classification (ou formation des concepts), qui structure les données en résumés et les résumés en une hiérarchie. Ensuite, les possibilités de représentation des résumés ont été détaillées de façon à permettre la mise en relief des propriétés des résumés. L'extension d'un résumé ne propose qu'une liste des enregistrements décrits. Par opposition, l'intension d'un résumé est très informative grâce aux degrés et mesures attachés à un résumé. La connaissance qui peut être tirée d'un résumé a été étudiée en dernier lieu.

Le travail présenté dans la suite de ce document pourrait s'appliquer à toute structure arborescente possédant certaines des propriétés des résumés du modèle SAINTETIQ :

- les feuilles contiennent des références (identifiants ou pointeurs) vers les données décrites ;
- les nœuds ont chacun une représentation unique dans l'arbre ;
- l'arbre est organisé suivant une relation d'ordre partiel dénotant l'englobement des nœuds de niveau inférieur par les nœuds de niveau supérieur.

Outre l'organisation des résumés en arbre, ces propriétés sont en effet celles qui seront exploitées, aussi bien pour l'interrogation des résumés que pour l'indexation des données.

CHAPITRE 2

Algorithme d'interrogation des résumés

Introduction

Dans le chapitre précédent, il a été établi qu'un résumé donne un aperçu d'une relation $\mathsf{R} = (A_1, A_2, \ldots, A_n)$ sur laquelle sont construits des résumés. R contient des données structurées. Celles-ci sont organisées en enregistrements (dénommés dans la suite les tuples initiaux ou originaux) décrits par des attributs monovalués. Le processus de résumé consiste à décrire les enregistrements par des termes linguistiques, puis à regrouper les descriptions de tuples possédant des descriptions similaires de manière à exposer les concepts présents au sein des données. Grâce à la structuration des résumés en une hiérarchie, des résumés de granularités différentes sont obtenus.

L'interprétation d'un résumé particulier par un utilisateur est relativement aisée. Mais interpréter une hiérarchie entière l'est moins. En effet, le nombre de résumés dans une hiérarchie peut être conséquent. De plus, la formation des concepts dans le modèle SAINTETIQ crée des regroupements d'après des critères a priori complexes pour un utilisateur. Dans l'éventualité où les regroupements seraient ceux qu'aurait effectués l'utilisateur, un moyen d'utiliser les résumés reste nécessaire. Une application des résumés à l'issue de leur construction est donc envisagée dans ce chapitre.

L'exploitation d'une hiérarchie de résumés doit permettre d'en tirer plus de connaissance que celle offerte par l'interprétation des résumés qui la composent pris individuellement. Elle doit notamment offrir à l'utilisateur la possibilité de préciser ses critères de regroupement, ceux qui lui semblent intéressants pour un traitement particulier. L'algorithme d'interrogation proposé dans ce chapitre est un outil qui répond à ce besoin ; il se fonde sur la hiérarchie déjà construite pour dégager de nouveaux regroupements. Cet outil permettra de répondre à des pré-

occupations telles que :

Comment sont les individus qui sont « x » sur X ? (2.1)

où X est une projection de la relation R sur un ensemble ordonné d'attributs implicitement défini dans la requête ($X \subseteq$ R). Une telle requête spécifie les individus ou objets à prendre en compte par l'indication des caractéristiques x portant sur les attributs X.

Répondre à une requête consiste donc à parcourir la hiérarchie de résumés afin d'y trouver les résultats de la requête. Bien qu'il soit possible de retrouver les enregistrements (par leurs identifiants présents dans la hiérarchie), nous considérons pour l'instant que l'interrogation est limitée aux résumés, ce qui évite l'accès à la relation R. Les résultats du processus d'interrogation des résumés sont donc des ensembles de résumés, l'opération d'interrogation elle-même consistant en une description des résumés résultats. La possibilité d'obtenir des résultats constitués d'enregistrements est traitée dans le chapitre 5.

2.1 Formalisation

Cette section propose une formalisation de l'interrogation des résumés à l'aide de la logique propositionnelle. La formulation des requêtes, leur évaluation et la présentation des résultats seront ainsi abordées dans leurs principes généraux en dehors de toute implémentation. En raison de la variété des notions et des écritures correspondantes, les notations utilisées dans ce chapitre ainsi que dans la suite du document sont récapitulées ci-après :

- R : schéma de relation des données sur lesquelles une hiérarchie de résumés est construite ;
- $\mathcal{A}$: ensemble des attributs de R ;
- $\mathbf{n}$: cardinal de $\mathcal{A}$ $(\mathbf{n} = |\mathcal{A}|)$;
- A_i : élément générique de $\mathcal{A}$, représentant le i^e attribut de R ;
- t : tuple de la relation R ;
- ct : tuple candidat ; une représentation d'un tuple t ;
- z : résumé, nœud de la hiérarchie de résumés ;
- $Ch(z)$: ensemble des combinaisons de descripteurs possibles d'après l'intension de z ;
- D_{A_i} : domaine (qualitatif ou quantitatif) de l'attribut A_i ;
- $D^+_{A_i}$: domaine de A_i décrit par des étiquettes linguistiques $l^j_{\mathcal{A}_i}$ où $j = 1 \ldots |D^+_{A_i}|$;
- $\mathcal{F}(D^+_{A_i})$: ensemble des sous-ensembles flous que l'on peut construire sur $D^+_{A_i}$;
- φ : fonction de réécriture associant à t l'ensemble de ses différentes représentations d'après les connaissances de domaine ;

- $\mathcal{L}_{\mathcal{A}_i}(z)$: ensemble des étiquettes linguistiques $l^j_{\mathcal{A}_i}$ de $D^+_{A_i}$ utilisées dans l'intension de z ; $D^+_{A_i}$ n'apparaissant pas dans l'intension de z.
- $\mathcal{C}_i$: ensemble des étiquettes linguistique spécifiées comme critères sur l'attribut A_i ;
- R_z : ensemble des tuples candidats participant à l'extension de z.

2.1.1 Formulation des requêtes

Dans toute requête telle que (2.1), X définit implicitement une relation complémentaire Y par rapport à l'ensemble $\mathcal{A}$ des attributs de R. On considérera les attributs de X comme des attributs d'entrée et ceux de Y comme des attributs de sortie. Dans le cas général traité ici, les relations suivantes (où A est l'ensemble des attributs de R) sont établies :

1. $Y = A \setminus X$;
2. $X \neq \emptyset$.

Il est en effet peu probable qu'une requête ne précise aucun critère de sélection ($X = \emptyset$). Par conséquent Y, complémentaire de X, ne peut être égal à R. En revanche, si $X =$ R (c'est-à-dire $Y = \emptyset$), les caractéristiques sont données sur tous les attributs. L'interrogation n'a plus pour but de décrire les données puisqu'une description complète des données est déjà fournie par la requête ; elle est considérée comme un test d'existence. Il peut être utile de savoir si des données présentant des caractéristiques spécifiées existent. L'interrogation des résumés fournit une réponse aux requêtes de ce type.
Pour prendre en compte Y dans la requête (2.1), celle-ci peut être reformulée par :

Comment sont sur Y les individus qui sont x sur X ? (2.2)

Une requête s'exprime en spécifiant un tuple x de la relation X. Chaque valeur d'attribut $x.A_i$ explicite les caractères requis sur l'attribut A_i. Le champ $x.A_i$ est éventuellement multivalué, c'est-à-dire composée de plusieurs valeurs.

Définition 1 (Caractère requis) : *Un caractère requis pour un attribut A_i est une étiquette linguistique, élément de l'ensemble $D^+_{A_i}$, apparaissant dans une requête sur l'attribut A_i. On note $\mathcal{C}_i$, de cardinal m_i, l'ensemble des caractères requis pour A_i, défini en extension par : $\mathcal{C}_i = \{d^1_i, d^2_i, \ldots, d^{m_i}_i\}$. On admet, lorsque $m_i > 1$, qu'il n'y a pas de relation de préférence sur les éléments de $\mathcal{C}_i$. L'ensemble $\mathcal{C}_i$ est la caractérisation-attribut de l'attribut A_i.*

Il suit de la définition précédente que $\mathcal{C}_i = x.A_i$. Cependant, il nous reste à étendre cette définition de $x.A_i$ à x.

Définition 2 (Caractérisation initiale) : *La caractérisation initiale liée à une requête est l'ensemble des* $\mathbf{k}$ *caractères requis pour les attributs éléments de* X*. Cet ensemble, noté* $\mathcal{C}$*, est défini en extension par* $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\mathbf{k}}\}$.

Exemple 6 :
Soit la relation $\mathsf{R}_1 = (\texttt{épaisseur}, \texttt{dureté}, \texttt{température})$ *des matériaux (imaginaires) disponibles dans le catalogue d'une usine métallurgique. Ces matériaux sont de très longues plaques métalliques conditionnées en rouleaux de forme cylindrique. Les rouleaux sont vendus au volume à d'autres usines métallurgiques plus spécialisées pour transformation (alliage, laminage, ...). Ils sont décrits par l'épaisseur des plaques, un indice normalisé de dureté et leur température de fusion. Soit la requête suivante*[1] *:*

Comment sont les rouleaux malléables ? (2.3)

Il s'agit de décrire des rouleaux par leur épaisseur et leur température de fusion. On a $X = (\texttt{dureté})$ *et* $Y = (\texttt{épaisseur}, \texttt{température})$*. Il n'y a qu'un caractère requis (sur l'attribut* dureté*) :* $\mathcal{C}_1 = \mathcal{C}_{du} = \{doux\}$ *et la caractérisation initiale est* $\mathcal{C} = \{\ \{doux\}\ \}$.

Exemple 7 :
Considérons la même relation R_1 *avec comme requête :*

Comment sont les rouleaux malléables ou mous d'épaisseur moyenne ? (2.4)

L'objectif, pour cette requête, est de décrire des rouleaux par leur température de fusion. On a $X = (\texttt{dureté}, \texttt{épaisseur})$ *et* $Y = (\texttt{température})$*. Les caractères requis portent cette fois-ci sur deux attributs (*dureté *et* épaisseur*) et sont :*

- $\mathcal{C}_1 = \mathcal{C}_{du} = \{doux; mou\}$ *;*
- $\mathcal{C}_2 = \mathcal{C}_{ep} = \{moyen\}$.

La caractérisation initiale devient $\mathcal{C} = \{\ \{doux; mou\}, \{moyen\}\ \}$.

Pour une caractérisation initiale $\mathcal{C}$, le processus d'interrogation fournira, conformément à son rôle de description, une caractérisation $\widehat{\mathcal{C}}$ (de même forme que $\mathcal{C}$) sur la relation Y telle que tout tuple z résultat vérifiant :

$$\mathcal{L}_{\mathcal{A}_i}(z) \subseteq \mathcal{C}_i,\ 1 \leq i \leq \mathbf{k}\ , \tag{2.5}$$

[1] Le terme « malléables » se réfère aux matériaux décrits par *doux*.

vérifie également :

$$\mathcal{L}_{\mathcal{A}_j}(z) \subseteq \widehat{\mathcal{C}}_j,\ \mathbf{k} < j \leq \mathbf{n}\ , \tag{2.6}$$

où $\mathbf{k}$ est le nombre d'attributs sur lesquels portent une caractérisation et $\mathbf{n}$, le degré de la relation R. La caractérisation $\widehat{\mathcal{C}}$ décrit, sur les attributs de sortie, les résumés sélectionnés grâce à leur description sur les attributs d'entrée, fournie par $\mathcal{C}$.

Les équations (2.5) et (2.6) traduisent, d'une part, que les résumés qui constituent les résultats de l'interrogation répondent aux critères de sélection, et d'autre part, que la caractérisation $\widehat{\mathcal{C}}$ est composée des descripteurs des résumés sélectionnés sur les attributs de sortie. L'équation (2.6) sera satisfaite à condition que les résumés soient correctement sélectionnés par la condition (2.5). En effet, lorsque $\mathbf{k} > 1$, comment combiner les différentes conditions de sélection qui en sont déduites ? La section suivante attribue une sémantique aux conditions de sélections composées, en traduisant la requête en une proposition logique afin d'en obtenir une interprétation pertinente.

2.1.2 Connecteurs logiques

Intuitivement, la présence de plusieurs caractères requis sur le même attribut (par exemple, dans la requête (2.4)) dénote une alternative. Un résumé est sélectionné dès qu'il présente l'une des caractéristiques recherchées ; formellement, le connecteur intra-attribut est disjonctif. On peut donc associer à la caractérisation $\mathcal{C}_i = \{d_i^1, d_i^2, \ldots, d_i^{m_i}\}$ d'un attribut A_i une clause composée des éléments de $\mathcal{C}_i$:

$$Cl_i = \bigvee_{j=1}^{m_i} d_i^j\ . \tag{2.7}$$

La clause Cl_i traduit la disjonction sur les étiquettes linguistiques, considérées ici comme des variables logiques.

Par contre, l'existence de plusieurs caractérisations, comme dans la requête :

Comment sont les rouleaux mous d'épaisseur moyenne ? (2.8)

a une signification de présence simultanée des caractéristiques : le connecteur inter-attributs est conjonctif. La proposition $\mathcal{P}$, qui est la conjonction des clauses Cl_i, s'écrit alors :

$$\mathcal{P} = \bigwedge_{i=1}^{\mathbf{k}} (Cl_i)\ . \tag{2.9}$$

En présence de plusieurs caractères requis pour un attribut (requête (2.4)) ou de plusieurs attributs, la caractérisation initiale $\mathcal{C}$ fournie par une requête peut être traduite sous la forme d'une proposition logique comme suit :

Définition 3 (Proposition logique associée à une requête) : *Soit une caractérisation initiale* $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_\mathbf{k}\}$*, fournie par une requête, telle que :*

$$\begin{aligned} \mathcal{C}_1 &= \{d_1^1, d_1^2, \ldots, d_1^{m_1}\} \\ \mathcal{C}_2 &= \{d_2^1, d_2^2, \ldots, d_2^{m_2}\} \\ &\vdots \\ \mathcal{C}_\mathbf{k} &= \{d_\mathbf{k}^1, d_\mathbf{k}^2, \ldots, d_\mathbf{k}^{m_\mathbf{k}}\}. \end{aligned}$$

La proposition logique $\mathcal{P}$ *associée à* $\mathcal{C}$ *s'écrit sous forme normale conjonctive :*

$$\mathcal{P} = \bigwedge_{i=1}^{\mathbf{k}} (\bigvee_{j=1}^{m_i} d_i^j) \ , \tag{2.10}$$

l'opérateur $\bigvee$ *symbolisant la prise en compte de plusieurs caractères requis sur un même attribut et l'opérateur* $\bigwedge$*, celle de plusieurs attributs.*

Pour qu'un résumé z soit pris en compte dans le processus d'interrogation, il doit satisfaire la condition (2.5). Par conséquent, la proposition (2.9) est nécessairement satisfaite et z en est un modèle. On a alors $\mathcal{P}(z) = \texttt{VRAI}$ où $\mathcal{P}(z)$ est l'interprétation de $\mathcal{P}$ dans le contexte de z par la fonction de valuation suivante :

Définition 4 (Valuation des descripteurs) : *Dans le contexte d'un résumé* z*, la valuation d'un descripteur quelconque* d_i^j*, en tant que littéral d'une proposition logique* $\mathcal{P}$ *associée à une caractérisation initiale, est :*

- $v(d_i^j) = \texttt{VRAI}$ *si* $d_i^j \in \mathcal{L}_{\mathcal{A}_i}(z)$ *;*
- $v(d_i^j) = \texttt{FAUX}$ *si* $d_i^j \notin \mathcal{L}_{\mathcal{A}_i}(z)$.

Exemple 8 :
La requête (2.4) de l'exemple 7 a pour caractérisations : $\mathcal{C}_1 = \mathcal{C}_{du} = \{doux, mou\}$ *et* $\mathcal{C}_2 = \mathcal{C}_{ep} = \{moyen\}$*. Par conséquent, elle induit la proposition :* $\mathcal{P}_1 = (doux \vee mou) \wedge moyen$*.*

2.1.3 Langage d'interrogation

Une formulation littérale d'une requête comme dans (2.2) ou (2.3) est insuffisante pour qu'elle soit prise en compte par un outil informatique. Nous introduisons une opération de description, traduite par le mot-clé « `DESCRIBE` » afin de permettre une formulation similaire à celles du langage SQL.

Ce mot-clé correspond à la tâche effectuée par l'interrogation, la description. On fait maintenant apparaître, comme l'indiquait la requête (2.2), les attributs pour lesquels on recherche une caractérisation (l'ensemble Y) par le mot-clé `ON`. Les attributs requis, implicites dans une formulation littérale, sont rendus explicites dans une clause `WHERE`. La forme générale d'une requête est :

`DESCRIBE` [<table>] `ON` <liste d'attributs> `WHERE` <conditions de sélection>

Nous reprenons ci-dessous les requêtes sur R déjà rencontrées, suivies chacune de leur expression avec les mots-clés mentionnés ci-avant ; la table MATERIAUX étant implicitement la source de données interrogée.

Comment sont les rouleaux malléables ?
`DESCRIBE ON épaisseur, température WHERE dureté IN` (*doux*)
Comment sont les rouleaux mous d'épaisseur moyenne ?
`DESCRIBE ON température WHERE dureté IN` (*mou*) `AND épaisseur IN` (*moyen*)
Comment sont les rouleaux malléables ou mous d'épaisseur moyenne ?
`DESCRIBE ON température WHERE dureté IN` (*doux*, *mou*) `AND`
`épaisseur IN` (*moyen*)

Le mot-clé `ON` permet de réduire l'ensemble Y à certains attributs plutôt qu'aux attributs absents de la clause `WHERE` ; ceci pourrait, dans certains cas particuliers, mener à des réponses plus concises.

2.2 Évaluation des requêtes

Comme le rappelle Chris Date dans [49], l'évaluation d'une requête dans un système de gestion de bases de données comprend habituellement quatre étapes :

1. l'expression de la requête dans un format interne ;
2. la réduction de l'expression interne à une forme canonique ;
3. le choix des procédures d'exécution ;
4. la génération de plans de requête et le choix du meilleur plan.

Dans le cadre de l'interrogation des résumés, le format interne est représenté par la notation ensembliste des caractérisations. On peut dès lors considérer qu'une forme canonique est atteinte puisqu'il n'y a qu'une représentation en extension d'un ensemble fini. L'existence d'une seule opération dans l'interrogation, la caractérisation, rend inutiles les deux dernières étapes.

Cette section traite de la seule procédure d'exécution, c'est-à-dire de l'exploration de la hiérarchie de résumés, par laquelle les résumés résultats de l'interrogation sont sélectionnés. L'exploration de l'arbre des résumés est un parcours en profondeur préfixé, au cours duquel les nœuds visités sont soumis à un test de correspondance détaillé en section 2.2.2. Ce parcours est d'autant plus adapté qu'il nécessite moins d'espace mémoire qu'un parcours en largeur et que les hiérarchies de résumés mettent en évidence une hauteur très inférieure à la largeur maximale. La profondeur moyenne d'une feuille est en effet estimée à $\log_B L$ où B est le nombre moyen de fils pour un nœud et L est le nombre de feuilles de la hiérarchie [127]. De plus, les résumés produits par SAINTETIQ sont stockés dans des fichiers XML. Ils suivent donc un ordre préfixé, typique de la sérialisation de données arborescentes. Enfin, le parcours en profondeur permet de réaliser des coupures de branches qui contribuent largement à réduire le nombre de nœuds visités pendant la recherche.

Le parcours séquentiel de tous les résumés est une solution simple mais qui implique que tous les résumés sont visités. Aucune réduction de l'espace de recherche n'intervient dans ce contexte. L'option du parcours séquentiel n'est donc pas envisagée.

2.2.1 Proposition logique et sémantique des résumés

La traduction d'une caractérisation initiale $\mathcal{C}$ sous forme de proposition logique a pour but de simplifier l'interprétation de $\mathcal{C}$. Néanmoins, il n'est pas toujours trivial de décider si un résumé quelconque z participe à la réponse de la requête. En effet, la description intensionnelle de z offre une lecture immédiate par rapport au connecteur disjonctif ($\vee$) mais pas par rapport au connecteur conjonctif ($\wedge$) :

- Considérons un résumé z et une caractérisation initiale $\mathcal{C}$ tels que $z.A = \{d_1, d_2, d_3\}$ et $\mathcal{C}_1 = \{d_1, d_2\}$. L'ensemble $\mathcal{L}_A(z)$ des descripteurs de z sur l'attribut A est $\mathcal{L}_A(z) = \{d_1, d_2, d_3\}$. Comme d_1 appartient à $\mathcal{L}_A(z)$, la clause $C\,l_1 = d_1 \vee d_2$ est validée car la variable logique d_1 est valuée à `VRAI`. Il en est de même pour $d_2 \in \mathcal{L}_A(z)$.
- Considérons ensuite $\mathcal{C} = \{\ \{a_1, a_2\}, \{b_1, b_2\}\ \}$ et $z = \langle\{a_1, a_2, a_3\}, \{b_1, b_3\}, \ldots\rangle$. Les clauses $Cl_1 = a_1 \vee a_2$ et $Cl_2 = b_1 \vee b_2$ sont satisfaites par z et la proposition $\mathcal{P} = Cl_1 \wedge Cl_2$ l'est également. Cependant, on ne peut conclure que z participe à la réponse de la requête sans autre précaution. Dans le cas où son extension ne couvre que des tuples ayant l'une des formes ci-après, z ne devrait pas être sélectionné car aucun de ses tuples n'est un résultat valide :
  - $(a_1, b_3, \ldots)$
  - $(a_2, b_3, \ldots)$

– $(a_3, b_1, \ldots)$
– $(a_3, b_3, \ldots)$

Ceci montre bien que l'interprétation d'une intension de résumé n'est pas triviale vis-à-vis de la conjonction.

D'autre part, une intension multivaluée autorise différentes combinaisons de descripteurs sur les attributs concernés. Notons $Ch(z)$ l'ensemble des combinaisons possibles d'après l'intension de z. Cet ensemble correspond au produit cartésien des ensembles de descripteurs dans l'intension de z. Par exemple, les combinaisons de descripteurs suivantes sont des valeurs valides pour les feuilles de $z = \langle\{a_1, a_2, a_3\}, \{b_1, b_2\}, \ldots\rangle$ restreintes aux deux premiers attributs : $\{a_1, b_1\}$, $\{a_1, b_2\}$, $\{a_2, b_1\}$, $\{a_2, b_2\}$, $\{a_3, b_1\}$, $\{a_3, b_2\}$. En conséquence, considérer z comme un résultat dès que la proposition liée à la requête est validée reviendrait à supposer que toutes les combinaisons sont représentées dans la hiérarchie de résumés, ce qui n'est pas garanti par le modèle SAINTETIQ.

On constate ainsi qu'un résumé z satisfaisant la proposition $\mathcal{P}$ dérivée d'une requête, peut ne pas être un résultat valide de la requête. Il suffit en effet que les combinaisons de descripteurs induites par l'intension du résumé soient plus générales que celles issues des caractérisations de la requête :

$$\prod_{i=1}^{\mathbf{k}} \mathcal{C}_i \subseteq Ch(z) \tag{2.11}$$

En conclusion, la satisfaction de la proposition $\mathcal{P}$ par un résumé n'implique pas que le résumé considéré est un résultat valide de la requête. Ce problème d'ambigüité est similaire au problème d'interprétation des intensions de résumés, déjà évoqué en section 1.6.

2.2.2 Conditions de sélection d'un résumé

Un résumé z participe à la réponse à une requête lorsqu'il satisfait la caractérisation initiale. En première approximation, ceci se traduit par le fait que z valide la proposition logique $\mathcal{P}(z)$ en permettant aux étiquettes, utilisées comme variables booléennes, d'être valuées positivement. Ces étiquettes linguistiques décrivent les données de z et on a alors $\mathcal{P}(z) = \texttt{VRAI}$. Cependant, la section 2.2.1 montre que la satisfaction de la proposition n'est pas suffisante pour garantir un résultat correct. Néanmoins, la proposition logique permet d'écarter les résumés qui n'offrent pas de correspondance avec la caractérisation initiale $\mathcal{C}$. Pour un tel résumé z', la proposition $\mathcal{P}$ n'est pas satisfaite : $\mathcal{P}(z') = \texttt{FAUX}$.

Dans la suite de cette section, la formulation ensembliste des requêtes fournie par la caractérisation initiale (voir la section 2.1.1 pour une définition) est utilisée afin de réaliser des tests de correspondance où le problème évoqué dans la section 2.2.1 précédente ne se pose pas. Il ne s'agit plus d'évaluer une proposition logique dans le cadre d'un résumé, mais de déterminer la situation des deux ensembles de descripteurs mis en jeu, l'un par rapport à l'autre. Ces ensembles sont d'une part, les caractères requis fournis par une requête et, d'autre part, les descripteurs extraits d'intensions de résumés. L'équivalence qui existe entre interprétation logique et opérations ensemblistes permet de traduire la conjonction logique par l'intersection et la disjonction par l'union.

Évaluer les conditions de sélection sur un résumé z revient donc à mettre en rapport l'ensemble $\mathcal{L}_{\mathcal{A}_i}(z)$ de ses descripteurs sur l'attribut $\mathcal{A}_i$ et la caractérisation-attribut $\mathcal{C}_i$ sur le même attribut $\mathcal{A}_i$. Bien sûr, tous les attributs présents dans la requête font l'objet de ce test de correspondance.

La figure 2.1 illustre les cinq différentes situations découlant de la comparaison de deux ensembles, en l'occurrence $\mathcal{L}_{\mathcal{A}_i}(z)$ et $\mathcal{C}_i$:

1. la disjonction des deux ensembles (fig. 2.1-a) ;
2. l'égalité des deux ensembles (fig. 2.1-b) ;
3. l'inclusion de $\mathcal{L}_{\mathcal{A}_i}(z)$ dans $\mathcal{C}_i$ (fig. 2.1-c) ;
4. l'inclusion de $\mathcal{C}_i$ dans $\mathcal{L}_{\mathcal{A}_i}(z)$ (fig. 2.1-d) ;
5. l'intersection partielle, indiquant une intersection non vide distincte de l'inclusion et de l'égalité (fig. 2.1-e).

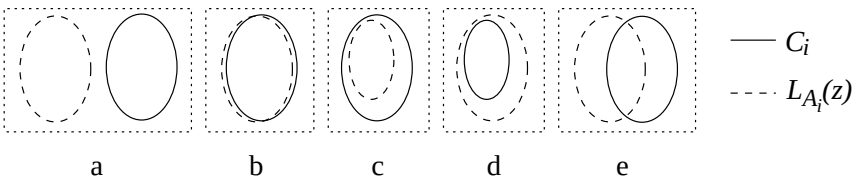


Figure 2.1 – Comparaison des ensembles de descripteurs $\mathcal{L}_{\mathcal{A}_i}(z)$ et $\mathcal{C}_i$

La situation relative des ensembles $\mathcal{L}_{\mathcal{A}_i}(z)$ et $\mathcal{C}_i$, rapportée à l'ensemble des attributs de la requête en cours d'évaluation, permet de prendre une décision quant au statut de z en tant que résultat de la requête. On distingue trois cas, détaillés ci-dessous.

Cas 1 (correspondance nulle). Il existe au moins un attribut pour lequel la clause de sélection de la requête n'est pas satisfaite : $\mathcal{P}(z) =$ `FAUX`. Le résumé n'est pas un résultat et le sous-arbre de racine z ne sera pas exploré.

Exemple 9 :
Soit une requête $\mathcal{Q}_1$ = `DESCRIBE ON température, dureté WHERE épaisseur IN` *(fin, mince). La caractérisation sur l'épaisseur (*$\mathcal{C}_{\texttt{ep}} = \{$*fin, mince*$\}$*) n'est pas satisfaite par* z_1 *tel que* $z_1.\texttt{ep} = \{1.0/$*moyen* $+ 1.0/$*épais*$\}$.
De même, pour une requête $\mathcal{Q}_2$ = `DESCRIBE ON température WHERE dureté IN` *(doux, mou)* `AND épaisseur IN` *(moyen),* $z_2 = \langle 1.0/$*moyen*$, 0.7/$*dur*$, 1.0/$*modéré*$\rangle$ *n'est pas un résultat car* $z_2.\texttt{dureté} = \{0.7/$*dur*$\}$ *ne présente aucune des caractéristiques recherchées.*

La section 2.2.1 montre que la sélection des résumés doit se faire sur un autre critère lorsque $\mathcal{P}(z) =$ `VRAI`. En effet, la satisfaction de la proposition par un résumé donné ne suffit pas à déterminer que ce dernier est un résultat valide de la requête. Pour lever l'ambigüité, on utilisera la relation ensembliste liant $\mathcal{C}_i$ et $\mathcal{L}_{\mathcal{A}_i}(z)$ car le problème d'ambigüité décrit par l'expression (2.11) nécessite que l'on ait, sur au moins un attribut, un descripteur de $z.\mathcal{A}_i$ qui n'appartienne pas à $\mathcal{C}_i$ (voir le cas 3 plus loin).

Cas 2 (correspondance exacte). Pour tous les attributs de la requête, le résumé présente uniquement des caractéristiques recherchées. Ce cas de figure ne fait intervenir que des situations relatives d'égalité ou d'inclusion (fig. 2.1-b et 2.1-c)) : $\forall\, i \in \{1, \ldots, \mathbf{k}\},\ \mathcal{L}_{\mathcal{A}_i}(z) \subseteq \mathcal{C}_i$. Ici, toutes les combinaisons de descripteurs du résumé sont des résultats valides de la requête. On a donc $Ch(z) \subseteq \prod_{i=1}^{\mathbf{k}} \mathcal{C}_i$. Le résumé z est un résultat ainsi que tous les éventuels nœuds du sous-arbre de racine z. L'exploration de ce sous-arbre n'est donc pas nécessaire. Un exemple pour ce cas est celui de la requête $\mathcal{Q}_3$ = `DESCRIBE ON température, dureté WHERE épaisseur IN` (*fin*, *mince*, *moyen*). La caractérisation initiale est constitué de $\mathcal{C}_{\texttt{ep}} = \{$*fin*$, $*mince*$, $*moyen*$\}$ et le résumé z_3 tel que $z_3.\texttt{ep} = \{0.6/$*fin* $+ 0.8/$*mince*$\}$ en est un résultat.

Cas 3 (correspondance par excès). Ce cas correspond à l'expression (2.11) : la proposition logique associée à la requête est satisfaite par un résumé z, mais il reste possible qu'aucun résultat ne soit trouvé dans le sous-arbre de racine z. Cette situation se présente typiquement lorsque le résumé dispose, sur un ou plusieurs attributs, de plus de descripteurs que ceux de la requête : $\exists\, i, \exists\, d \in \mathcal{L}_{\mathcal{A}_i}(z)\ /\ d \notin \mathcal{C}_i$. Par exemple, pour $\mathcal{Q}_1$ = `DESCRIBE ON température, dureté WHERE épaisseur IN` (*fin*, *mince*), la caractérisation $\mathcal{C}_{\texttt{ep}} = \{$*fin*$, $*mince*$\}$ est satisfaite par $z_4.\texttt{ep} = \{1.0/$*fin* $+ 0.8/$*mince* $+ 0.4/$*moyen*$\}$.

Dans une correspondance de ce type, il existe dans z au moins un descripteur (*moyen* dans l'exemple) en plus des caractères requis. $\mathcal{L}_{\mathcal{A}_i}(z)$ est un sur-ensemble de $\mathcal{C}_i$ (fig. 2.1-d) ou tient

compte d'autres caractères que ceux recherchés (fig. 2.1-e). Si z participait à la réponse, il ne serait pas possible d'assurer que celle-ci reflète *seulement* les caractères requis. Afin de ne retenir que les résumés adéquats, l'exploration du sous-arbre de racine z s'impose donc.

2.2.3 Algorithme de recherche

La sélection des résumés est effectuée par la fonction *Recherche* présentée dans l'algorithme 1. Cette fonction récursive effectue la sélection des résumés résultats dans un sous-arbre. Elle prend en entrée un résumé z, racine du sous-arbre, et la caractérisation $\mathcal{C}$, forme canonique de la requête exprimée sous forme d'ensembles de descripteurs. Elle retourne une liste indifférenciée de résumés notée L_{res}. Au fil du parcours, le contenu de la liste est mis à jour de manière à refléter les résultats déjà rencontrés.

L'algorithme implémente le parcours en profondeur et les conditions de sélection, discutées en section 2.2.2, applicables à chaque résumé visité. Au moment de l'examen d'un nœud z, l'exploration du sous-arbre peut ne pas se poursuivre au-delà de z. Ceci se justifie par une correspondance nulle ou une correspondance exacte (cas 3 de la section 2.2.2). Dans ce dernier cas, z est rajouté à L_{res}. Lorsqu'une correspondance par excès survient, l'exploration se poursuit par un appel récursif de la fonction, sans modification de la liste de résultats. Nous admettons

Algorithme 1 Fonction Recherche(z, $\mathcal{C}$)

```
L_res ← <> {la liste est vide}
si Corr(z, C) = excès alors
   pour chaque nœud fils z_fils de z faire
      L_res ← L_res + Recherche(z_fils, C)
   fin pour
sinon
   si Corr(z, C) = exacte alors
      ajouter(z, L_res)
   fin si
fin si
retourner L_res
```

les hypothèses suivantes :

- la fonction $Corr$, qui représente l'évaluation des conditions de sélection, prend ses valeurs dans l'ensemble $\{\textit{nulle}, \textit{exacte}, \textit{excès}\}$;

- l'opérateur binaire « + » est défini sur les listes et réalise une concaténation des listes opérandes ;
- la fonction *ajouter* est le constructeur classique réalisant l'ajout d'un élément dans une liste de type adéquat ;
- L_{res} est une variable locale à la fonction *Recherche*.

À propos de la complexité. La sensibilité à l'ordre des hiérarchies de résumés induit des hiérarchies de résumés supposées différentes pour le même jeu de données. Le parcours de ces hiérarchies, pour un même ensemble de données et une même requête est susceptible de suivre des chemins différents. De plus, pour une même hiérarchie de résumés, des requêtes différentes déterminent des sélections supposées différentes. Ici encore, il est peu probable que l'exploration conduise à visiter les mêmes nœuds. Pour ces raisons, il est difficile de fournir une complexité moyenne. Nous considérons le pire des cas, qui correspond à celui où toutes les feuilles de la hiérarchie de résumés sont sélectionnées. Tous les nœuds sont donc visités. Il faut cependant noter que cette hypothèse très rarement effective en pratique, notamment en raison des coupures de branche dont le rôle est justement d'éviter une recherche exhaustive.

Puisque la fonction est récursive, et que chaque pas correspond à une progression d'un niveau dans la hiérarchie, la complexité spatiale de l'exploration est celle d'un parcours en profondeur récursif. Elle est de l'ordre de h, la hauteur de la hiérarchie. L'espace mémoire utilisé comprend la liste des résumés sélectionnés, le résumé racine du sous-arbre, la caractérisation initiale et un résumé fils courant.

En considérant la fonction de sélection elle-même comme opération élémentaire, le pire des cas est celui où la fonction est invoquée pour chaque nœud de la hiérarchie. La complexité temporelle de l'exploration est donc de l'ordre de n, le nombre de résumés dans la hiérarchie.

2.2.4 Formulation des résultats

Une fois l'identification des nœuds désignés par la requête effectuée, la réponse à la requête est une liste de résumés. Une opération de classification est nécessaire car des résumés distincts peuvent répondre à la requête de manières différentes. Rappelons qu'une hiérarchie de résumés est construite par une classification de tuples candidats, représentations linguistiques des données. Il n'est pas question de reprendre ici un processus de classification mais de s'en inspirer pour proposer une formulation simple des résultats autre qu'une liste. Ainsi, une requête $\mathcal{Q}_5$ = `DESCRIBE ON température, dureté WHERE épaisseur IN` (*fin*, *moyen*) présente éventuellement parmi ses résultats des résumés dont l'épaisseur est décrite par *fin*, d'autres par *moyen*,

et encore d'autres par les deux étiquettes *fin* et *moyen*. Le principe de la formulation présentée ci-dessous est de regrouper les résultats en fonction de leurs caractéristiques vis-à-vis de la requête ; les résumés décrits par *fin* fourniront une caractérisation distincte de celle des résumés décrits par *moyen*. Cette méthode présente plusieurs avantages :

- l'existence de certaines combinaisons de descripteurs requis au sein des données peut être facilement vérifiée ;
- les catégories de résumés résultats sont différenciées : on sait précisément à quels résumés est due la présence d'un descripteur en sortie ;
- la présentation des résultats est synthétique et permet, si nécessaire, d'avoir une unique description globale.

En l'occurrence, le regroupement des résultats s'effectuera en fonction de l'interprétation, au sens de la logique propositionnelle, qui valide la proposition $\mathcal{P}$. La requête traduite en formule logique s'écrit sous forme normale conjonctive par : $\mathcal{P} = \bigwedge_{i=1}^{\mathbf{k}} Cl_i$ avec des clauses $Cl_i = \bigvee_{j=1}^{m_i} d_i^j$.

Étant donné qu'une clause Cl_i est satisfaite dès que l'une de ses variables est valuée positivement, il existe $2^{m_i} - 1$ interprétations validant toute clause logique de m_i variables. Par exemple, chacun des ensembles $\{\textit{fin}\}$, $\{\textit{moyen}\}$, $\{\textit{fin}, \textit{moyen}\}$ fournit une interprétation de $Cl_{\texttt{ep}} = (\textit{fin} \vee \textit{moyen})$. En effet, une valuation positive des variables de l'un de ces trois ensembles conduit à une satisfaction de $Cl_{\texttt{ep}}$.

Chacune de ces $2^{m_i} - 1$ interprétations est une manière dont un résumé z satisfait $\mathcal{C}_i$. À l'échelle de la proposition $\mathcal{P}$ plutôt que de la clause, le nombre N d'interprétations est :

$$N = \prod_{i=1}^{\mathbf{k}} (2^{m_i} - 1) \ . \tag{2.12}$$

Puisqu'il existe a priori plusieurs interprétations de la proposition logique relative à la requête, il existe autant de *manières* de répondre à la requête. La description réalisée par l'interrogation est en réalité un ensemble $\widehat{\mathcal{C}}_{out}$ de caractérisations. À l'image de la caractérisation initiale fournie par la requête, la caractérisation $\widehat{\mathcal{C}}_{out}$ décrit les résumés sélectionnés, mais sur l'ensemble Y des attributs de sortie (voir section 2.1.1) : $\widehat{\mathcal{C}}_{out} = \{\widehat{\mathcal{C}}_{k+1}, \ldots, \widehat{\mathcal{C}}_{\mathbf{n}}\}$.

Définition 5 (Caractérisation résultat) : *Soit E l'ensemble des résumés correspondant à une caractérisation initiale $\mathcal{C}$ et satisfaisant $\mathcal{P}$ par la même interprétation logique. La caractérisation résultat sur la relation Y s'exprime par :*

$$\widehat{\mathcal{C}}_j = \bigcup_{j=k+1}^{\mathbf{n}} \mathcal{L}_{A_j}(z), \quad z \in E \ ,$$

avec $\mathbf{n}$ *le degré de la relation* R*,* A_1 *à* $A_{\mathbf{k}}$ *les attributs d'entrée et* $A_{\mathbf{k}+1}$ *à* $A_{\mathbf{n}}$*, les attributs de sortie.*

Exemple 10 :
Reprenons la requête (2.4)

Comment sont les rouleaux malléables ou mous d'épaisseur moyenne ?

et considérons qu'elle est adressée à la relation R_1 *=(*épaisseur*,* dureté*,* température*) de l'exemple 6. On a :*

- $X = (\texttt{dureté}, \texttt{épaisseur})$
- $Y = (\texttt{température})$
- $\mathcal{C} = \{\ \{\textit{moyen}\}, \{\textit{doux}; \textit{mou}\}\ \}$
- $\mathcal{P} = \textit{moyen} \wedge (\textit{doux} \vee \textit{mou})$

Les résumés suivants sont sélectionnés :

- $z_1 = \langle \{0.4/\textit{moyen}\}, \{1.0/\textit{doux}\}, \{0.6/\textit{modéré}\} \rangle$
- $z_2 = \langle \{0.8/\textit{moyen}\}, \{0.8/\textit{doux}\}, \{1.0/\textit{normale}\} \rangle$
- $z_3 = \langle \{1.0/\textit{moyen}\}, \{0.8/\textit{mou}\}, \{1.0/\textit{bas}\} \rangle$
- $z_4 = \langle \{1.0/\textit{moyen}\}, \{0.8/\textit{mou} + 0.4/\textit{doux}\}, \{1.0/\textit{modéré}\} \rangle$
- $z_5 = \langle \{1.0/\textit{moyen}\}, \{0.7/\textit{doux}\}, \{1.0/\textit{normale} + 0.8/\textit{modéré}\} \rangle$
- $z_6 = \langle \{0.5/\textit{moyen}\}, \{0.5/\textit{mou} + 0.7/\textit{doux}\}, \{1.0/\textit{normale} + 0.8/\textit{modéré}\} \rangle$

D'après l'équation (2.12), la proposition $\mathcal{P}$ *a 3 interprétations différentes :*

- $\{\textit{moyen}, \textit{doux}\}$ *associée aux résumés* $\{z_1, z_2, z_5\}$
- $\{\textit{moyen}, \textit{mou}\}$ *associée à* $\{z_3\}$
- $\{\textit{moyen}, \textit{doux}, \textit{mou}\}$ *associée à* $\{z_4, z_6\}$

Ces interprétations donnent respectivement les résultats suivants :

- $\widehat{\mathcal{C}}^1 = \{\ \{\textit{modéré}, \textit{normale}\}\ \}$
- $\widehat{\mathcal{C}}^2 = \{\ \{\textit{bas}\}\ \}$
- $\widehat{\mathcal{C}}^3 = \{\ \{\textit{modéré}, \textit{normale}\}\ \}$.

Exemple 11 :
Si la requête était :

Comment sont les rouleaux d'épaisseur moyenne ?

on aurait :

- $X = (\texttt{épaisseur})$
- $Y = (\texttt{dureté}, \texttt{température})$

- $\mathcal{C} = \{ \{moyen\} \}$
- $\mathcal{P} = moyen$

Et si les mêmes résumés étaient sélectionnés, c'est-à-dire $z_1, \ldots, z_6$, on aurait comme seul résultat (puisque $\mathcal{P}$ n'a qu'une interprétation) :

- $\widehat{\mathcal{C}} = \{ \{doux, mou\}, \{modéré, normale, bas\} \}$.

Afin de faciliter la lecture des résultats, les résumés présentant les mêmes ensembles de descripteurs pour tous les attributs requis peuvent fournir une liste de descripteurs par attribut :

- fin, mou ⇒ froid ;
- fin, doux ⇒ froid, bas ;

2.3 Conclusion

Ce chapitre a abordé l'interrogation des résumés, c'est-à-dire la sélection de résumés au sein d'une hiérarchie sur la base de critères spécifiés par une requête. Les éléments formels du processus d'interrogation ont été décrits de manière indépendante d'une implémentation spécifique. C'est ainsi qu'une forme canonique des requêtes, correspondant à une représentation ensembliste des critères de requêtes, a été proposée. Cette forme canonique, qui reprend les descripteurs linguistiques de la requête dans des ensembles de caractéristiques requises, est comparée à chaque résumé pendant l'exploration de la hiérarchie de résumés afin de déterminer si le résumé considéré répond à la requête. Le résultat de cette comparaison détermine en outre la poursuite de l'exploration et permet, le cas échéant, d'élaguer l'espace de recherche (l'arbre des résumés).

Un algorithme de sélection de résumés, objet de publications [141, 143], a également été proposé. Cet algorithme, premier outil d'exploitation des hiérarchies de résumés, est un élément important de ce document puisqu'il sert de base aux traitements que nous aborderons ultérieurement. Il effectue le parcours en profondeur de la hiérarchie visée et réalise les tests de correspondance entre les résumés et la requête. Il devient ainsi possible de déterminer l'existence simultanée de caractéristiques, et plus généralement, de connaître les corrélations existant au sein des données.

CHAPITRE 3

Interrogation flexible

Introduction

Dans ce chapitre, l'interrogation flexible sera définie à partir de la façon dont elle est perçue dans la communauté scientifique. L'interrogation dite *flexible* adopte des formes diverses suivant le domaine que l'on considère (bases de données, recherche d'informations, data mining, ...) mais les résumés linguistiques de SAINTETIQ traitant de données structurées, notre champ d'étude sera limité aux bases de données relationnelles. Puis, une section sera consacrée à la notion de « préférence », notion centrale de l'interrogation flexible, sous ses diverses formes. Ensuite, quelques systèmes d'interrogation flexible, concrétisant les généralités précédemment mises en évidence, seront présentés.

Ce qu'est l'interrogation flexible en bases de données D'après Bosc, Motro et Pasi [23], un système d'interrogation est dit flexible s'il « permet d'exprimer des préférences de manière à ordonner les éléments plus ou moins acceptables qui auront été trouvés en fonction de leur adéquation à la requête ». De cette première acception, on déduit que :

- les préférences, explicitement définies ou non, prennent une part importante dans la perception actuelle de l'interrogation flexible (voir également [22, 24, 51, 120]) ;
- il est admis que les résultats d'une requête ne lui correspondront pas dans le sens le plus strict possible ;
- la notion d'adéquation, abordée en section 3.1.4, est exprimée par un degré de pertinence des résultats (ou de satisfaction de l'utilisateur) variable suivant l'interprétation faite des préférences et la nature des données interrogées.

La flexibilité peut aussi être « la capacité d'un système à répondre à une requête de manière coopérative » [23]. Cependant, cette perception des systèmes flexibles semble avoir une portée relativement faible, essentiellement limitée aux bases de données déductives (voir section 4.3.1). Enfin, un système est également dit flexible en raison de la nature des données traitées. Il s'agit en particulier des données *mal-connues* [19, 22].

En définitive, le caractère flexible de l'interrogation de bases de données désigne une variation autour du processus habituel d'interrogation (expression d'une requête dans un langage de manipulation de données structurées – habituellement, SQL – puis évaluation booléenne de la requête et enfin production d'une liste indifférenciée de tuples en tant que réponse), notamment lorsque les données prises en compte sont de nature « non-classique », par exemple, les données incomplètes, mal connues, issues de la théorie des probabilités, etc.

Objectifs Le *paradigme* de l'interrogation flexible décrit par Bosc *et al.* dans [23] assimile un système flexible à un expert humain servant d'intermédiaire entre l'utilisateur et le système d'information. Suivant cette hypothèse, l'expert serait capable d'analyser les demandes des utilisateurs, de déterminer leurs besoins en termes d'information, de localiser les sources d'information adéquates, d'en extraire les éléments de réponse plus ou moins pertinents et de les ordonner. Cependant, il est évident qu'il existe une certaine différence entre l'utilisateur et l'expert ; ce dernier ne peut totalement cerner les besoins de l'utilisateur. L'expert ne peut faire qu'une *estimation* des éléments pertinents. La qualité du système d'interrogation sera largement influencée par celle de la mise en correspondance entre, d'une part, l'expression de la requête et, d'autre part, les besoins à satisfaire, déduits de la requête, que le système adopte comme objectifs.

Deux éléments importants interviennent dans cette mise en correspondance : l'interprétation des besoins de l'utilisateur et l'interprétation des données au regard des besoins. L'interrogation flexible a comme contrainte la nécessité de prendre en compte ces deux transformations du mieux possible. Pour ce faire, la flexibilité a deux objectifs.

Premièrement, elle vise à faciliter l'expression des besoins en termes de clarté, de précision, de concision ou de proximité par rapport au langage naturel [51, 118]. Par exemple, Larsen [96] pointe le fait que les langages de manipulation de données – nommément CCL (Common Command Language) et SQL (Structured Query Language) – tiennent plus compte du système de traitement (bases de données et systèmes informatiques) que de l'utilisateur. Celui-ci doit s'immerger dans les arcanes du langage pour exprimer clairement ses besoins de manière à ce qu'ils soient fidèlement satisfaits. C'est une contrainte difficilement imposable à l'utilisateur moyen, en raison de l'investissement en temps et des connaissances préalables qui seraient requis. Larsen propose donc que le langage d'interrogation soit « plus naturel et plus intuitif ».

Toujours du côté de l'utilisateur, l'interrogation flexible vise aussi une meilleure présentation des réponses de requêtes. Habituellement, le langage SQL présente une liste de résultats. Dans le cas où cette liste est importante (plus de quelques enregistrements), l'utilisateur s'y

perd. Certains travaux [16, 50, 64, 65, 69] portent sur une expression intelligible des résultats afin d'en permettre une meilleure exploitation. Classés en général parmi les systèmes à caractère coopératif, ils aident l'utilisateur à interpréter les résultats ou les expriment de manière intensionnelle.

Deuxièmement, la flexibilité vise à prendre en compte des données qui auraient, autrement, été exclues du processus habituel d'interrogation parce que mal définies, mal connues, incomplètes, ou hors (mais proches) des critères de sélection. Les travaux sur la *fuzzification* des bases de données [24, 27] ou du langage d'interrogation [24] veulent englober les données imparfaites, se montrant ainsi plus proches de la réalité reflétée par une base de données.

Conventions Dans la suite de ce document, on considère qu'un système d'interrogation traite les requêtes qui lui sont adressées en vue de fournir une « réponse ». La réception d'une requête déclenche une procédure de recherche (également désignée par « la recherche » ou « la sélection ») qui examine tout ou partie de l'espace de recherche (une base de données ou une table de données) pour trouver des résultats. À chaque requête en entrée correspond une réponse en sortie. Cependant, une réponse est constituée de un ou plusieurs résultats. Dans le cadre précis des bases de données, un résultat est un enregistrement sélectionné lors de la recherche.

On supposera par ailleurs que, dans une requête, des critères de sélection sont spécifiés pour des attributs dits « d'entrée » ; un critère, qu'il soit simple ou complexe, sera ainsi toujours associé à un attribut. Ainsi, une requête SQL dont la clause de sélection est :

```
WHERE age < 25 AND dpt = 16
```

dispose de deux critères simples sur les attributs `age` et `dpt`, tandis qu'une requête où la clause de sélection est :

```
WHERE age > 18 AND age < 25
```

a un critère, sur l'attribut `age`, complexe car faisant intervenir plus d'une valeur.

On admettra également que la réponse à une requête fournit des informations sur les attributs « de sortie », représentés dans une requête SQL par la clause SELECT, par exemple, *nom* et *salaire* dans :

```
SELECT nom, salaire FROM Employés WHERE age < 25 AND dpt = 16;
```

3.1 Préférences

L'introduction des préférences en interrogation de bases de données vise en premier lieu à assouplir la « manière rigide dont les caractéristiques des données recherchées doivent être spécifiées » [94]. Cette rigidité conduit à écarter systématiquement les objets aux valeurs d'attribut non idéales. Dans le cas où aucun objet (aucun enregistrement) ne répond à ces caractéristiques, il est néanmoins possible, dans certaines applications, d'accepter des objets répondant à de *moins bonnes* caractéristiques au regard des critères de recherche.

Une requête avec préférences définit, parfois de manière implicite, deux éléments :

- une hiérarchie sur les critères de sélection ;
- un ordre de substitution portant sur les propriétés ou qualités recherchées. Cet ordre est identique à celui observé parmi les résultats acceptables.

Pour traduire l'ordre de substitution, l'évaluation d'une requête à préférences peut être assimilée à l'évaluation d'une première requête débouchant sur celle d'une deuxième requête si la réponse à la première est vide. Si cette deuxième requête ne fournit pas de résultat, une troisième requête entre en jeu et ainsi de suite jusqu'à ce qu'une réponse non vide soit trouvée ou jusqu'à ce que toutes les substitutions aient été épuisées. Ce comportement est identique à celui d'un utilisateur qui, face à un système d'information classique, pose des requêtes successives le guidant progressivement vers les résultats recherchés.

Exprimer des préférences dans une requête permet de spécifier toutes les requêtes alternatives en même temps que la requête initiale. Dans le scénario ci-dessus, l'absence de résultat possédant les caractéristiques appropriées constitue un *critère de déclenchement* de la préférence. Les préférences de ce type (où les qualités recherchées sont rendues explicites), dites *préférences qualitatives*, sont exposées en section 3.1.2.1.

Une autre motivation de l'utilisation des préférences, mise en avant plus récemment par Hristidis *et al.* [83], tient au fait qu'il est impossible, dans un langage comme SQL, de spécifier l'importance relative d'une caractéristique par rapport à une autre (« pour moi, le confort d'une voiture est plus important que la puissance »). En effet, tous les critères de sélection dans une requête sont pris en compte de manière uniforme. Parce qu'il n'y a pas de notion de séquence ou de hiérarchie, l'évaluation de la requête ne peut être assimilée à l'évaluation, conditionnée par un critère de déclenchement (en général, l'échec de la requête précédente), d'une série de requêtes. Les préférences qualitatives ne peuvent donc plus s'appliquer ; ce cadre est celui des préférences quantitatives (traitées en section 3.1.2.2). Dans le même ordre d'idées, Bosc *et al.* [22] relèvent que, dans les systèmes classiques, il est également impossible de spécifier une préférence sur les valeurs d'attributs (« je préfère une teinte vive à une teinte pastel »).

L'utilité des préférences est de permettre l'acceptation de certains résultats lorsque des résultats, qui leur sont préférés, ne sont pas retrouvés dans la base de données. Plutôt que d'exécuter une recherche, de constater l'absence de résultats, et de relancer une recherche, les préférences permettent une *économie* de requêtes et donc de procédures de recherche, de temps de réponse (d'un point de vue global) et de calcul. De plus, le système d'information capable de prendre en compte les préférences des utilisateurs est perçu comme plus versatile et plus convivial, entre autres avantages [59].

Bien qu'aucun autre article n'en fasse mention, Lacroix et Lavency [94] indiquent une autre application pour les préférences : réduire le volume de la réponse en renforçant les critères en cas de nombre élevé de résultats. En effet, les préférences spécifiées dans une requête correspondent en général à un relâchement des critères de sélection en présence d'un nombre faible ou insuffisant de résultats. Dans ce cas, nous conjecturons que le comportement de renforcement des critères peut être obtenu de manière simple tout en gardant le cadre du relâchement des critères. La première modification porte sur le critère de déclenchement des préférences. Ce critère (l'absence de résultat dans le relâchement) deviendrait un nombre de résultats supérieur à un seuil spécifié. La deuxième modification consiste à choisir des caractéristiques de substitution plus contraignantes plutôt que des caractéristiques moins restrictives. Le scénario de prise en compte des préférences devient : « si la condition *cond* est remplie, appliquer la modification *modif* ». La réduction du nombre de résultats s'obtiendrait pour :

$$cond = nombre\ de\ résultats > seuil \quad .$$

Même en l'absence d'utilisation explicite du terme « préférence » par leurs auteurs, les techniques utilisées en interrogation flexible de bases de données définissent des préférences dès lors qu'un tri des résultats est effectué. Par exemple, celles basées sur les sous-ensembles flous ([24, 87, 117]) présentent en priorité les résultats correspondant aux noyaux des sous-ensembles flous. Ces sous-ensembles sont soit spécifiés dans les critères de sélection, soit dérivés par calcul.

La suite de cette section consacrée aux préférences traite de l'interprétation qui peut en être faite avant d'exposer une typologie des modèles de traitement de préférences. Elle aborde également les problèmes liés à ce traitement et l'adéquation des résultats vis-à-vis des requêtes.

3.1.1 Sémantique des expressions de préférences

Les indications fournies par une expression de préférence, notée $\succ$, sont explicites dans « je préfère les tables rondes aux tables carrées » : les objets de valeur *ronde* sur l'attribut `forme`

sont préférés à ceux de valeur *carrée*. Bien que cette expression soit la plus simple possible (une valeur sur un attribut) en matière de préférences, son interprétation n'est pas unique. En effet, les objets extraits par une requête sont également décrits par d'autres attributs. La question de l'interprétation des préférences se résume à la manière dont ces autres attributs sont pris en compte dans un classement des résultats.

Soient t et t' deux enregistrements d'une relation. Soit $\succ$ une relation spécifiant une préférence d'une valeur v sur une valeur v' d'un attribut A_k. Brafman et Domshlak [31] mettent en évidence deux interprétations possibles menant à la précédence de t sur t' :

1. $t.A_k = v \succ t'.A_k = v' \quad \Leftrightarrow \quad t \succ t'$
2. $(t.A_k = v \succ t'.A_k = v') \wedge (\forall j \ / j \neq k, t.A_j = t'.A_j) \quad \Rightarrow \quad t \succ t'$

Dans la première interprétation, où la sémantique est dite totalitaire, les autres attributs sont simplement ignorés : la précédence s'applique entre deux objets dès que leurs valeurs sont celles spécifiées dans l'expression de préférence.

Exemple 12 :

Considérons les enregistrements suivants d'une table de véhicules d'occasion suivant le schéma de relation (marque, couleur, ...) et une relation de préférence sur la couleur rouge $\succ$ noir *:*

- $t_{sr} = \langle$*Smart, rouge, ...*$\rangle$
- $t_{sn} = \langle$*Smart, noir, ...*$\rangle$
- $t_{mr} = \langle$*Mini, rouge, ...*$\rangle$
- $t_{mn} = \langle$*Mini, noir, ...*$\rangle$

Préférer les voitures rouges aux voitures noires implique que toutes les voitures rouges sont mieux placées *que toutes les voitures noires :* $t_{sr} \succ t_{sn}$, $t_{sr} \succ t_{mn}$, $t_{mr} \succ t_{sn}$ *et* $t_{mr} \succ t_{mn}$. *La question de la relation entre* t_{sr} *et* t_{mr} *d'une part, et* t_{sn} *et* t_{mn} *d'autre part, ne se pose pas en l'absence d'une autre relation de préférence, par exemple sur la marque de la voiture. L'important ici est que* t_{sr} *et* t_{mr} *soient clairement identifiés comme meilleurs que* t_{sn} *et* t_{mn}.

La seconde interprétation correspond à la sémantique *ceteris paribus*, cette expression signifiant « toutes autres choses étant égales » ou « toutes choses égales par ailleurs ». La précédence n'intervient qu'à la condition que les valeurs des objets sur les autres attributs soient égales. Ce deuxième cas d'interprétation étant plus restrictif que le premier, le nombre d'objets concernés par la précédence est plus faible car les objets incomparables sont plus nombreux.

Exemple 13 :

En l'occurrence, la même préférence du rouge par rapport au noir conduit au fait qu'une Smart rouge est mieux classée qu'une Smart noire (car t_{sr}*.couleur* $\succ$ t_{sn}*.couleur et* t_{sr}*.marque* $=$ t_{sn}*.marque), de même qu'une Mini rouge est mieux classée qu'une Mini noire (*t_{mr}*.couleur* $\succ$

t_{mn}.couleur et t_{mr}.marque = t_{mn}.marque). Mais la Smart rouge n'est comparable avec aucune Mini, y compris la Mini rouge (car t_{sr}.marque $\neq$ t_{mn}.marque et t_{sr}.marque $\neq$ t_{mr}.marque) et vice-versa.

Les auteurs estiment que c'est le faible pouvoir discriminant de cette deuxième interprétation qui justifie son manque de succès auprès des chercheurs traitant des préférences en bases de données, et ce en dépit du fait qu'elle est plus conforme, d'après d'autres communautés des sciences humaines comme celle des philosophes [78], à l'intention d'un utilisateur.

3.1.2 Types de préférences

Cette section décrit les deux types de préférences que distingue Chomicki [35] en fonction de leur expression formelle. Une typologie différente, basée sur la nature des préférences (simple / complexe, numérique / non-numérique) peut être trouvée dans [90]. D'autres distinctions peuvent être effectuées entre les différentes propositions, suivant la technique d'implémentation [22] : sous-ensembles flous, distances, similarités, ...

3.1.2.1 Préférences qualitatives

Certains travaux [31, 34, 35, 67, 94] sont spécifiquement orientés vers l'expression explicite par l'utilisateur des préférences au sein des requêtes. Brafman et Domshlak se distinguent en traitant dans [31] le problème, rarement abordé dans le domaine des bases de données, de la sémantique des préférences. L'interprétation des préférences n'est en effet pas bien définie même lorsqu'un langage ad'hoc est utilisé comme dans l'exemple suivant tiré des travaux pionniers de Lacroix et Lavency [94] en 1987 :

```
select the versions of MAIN
 having STATUS = coded
from which prefer those
 having TARGET = 16
```

} clause de préférence

L'interprétation intuitive de la requête est la suivante : elle est d'abord évaluée sans tenir compte de la clause de préférence. Puis, cette dernière est prise en compte pour filtrer les résultats de la requête. Dans le cas où des objets satisfaisant la clause de préférence existent, la réponse fournie sera limitée à ces objets. Autrement, la réponse est inchangée.

Le caractère hiérarchique est plus perceptible dans ce type de préférences car il est explicitement exprimé dans le cas de préférences imbriquées. Par exemple, une requête comme la suivante recherche les objets satisfaisant toutes les préférences indiquées (p_1 et p_2).

```
select the instances of CONF
 having
  the version of MAIN
   having
    same TARGET as the version of PROCESS-DATA and
    same TARGET as the version of GET-DATA
 from which prefer those                                    (p1)
  having
   the version of MAIN having STATUS = tested
 from which prefer those                                    (p2)
  having
   the version of PROCESS-DATA having STATUS = tested
```

Si de tels objets existent, ils satisfont la clause $C_n = p_1 \wedge p_2 \wedge \ldots \wedge p_n$ et constituent la réponse. Autrement, la réponse comprend les objets répondant à la clause $C_{n-1} = p_1 \wedge p_2 \wedge \ldots \wedge p_{n-1}$, sinon à $C_{n-2} = p_1 \wedge p_2 \wedge \ldots \wedge p_{n-2}$ et ainsi de suite. L'expression des préférences est destinée à refléter ce caractère restrictif, mais pas exclusif : la réponse n'est vide que si aucune préférence n'est satisfaite.

Formellement, une préférence définit une relation binaire de précédence $\succ$ entre objets au regard de leur valeur sur un attribut précis (des expressions plus complexes pouvant faire intervenir plusieurs attributs). Dans une requête à préférences $\mathcal{Q}$ où m préférences $s_1, \ldots, s_m$ sont spécifiées, des relations $\succ_1, \succ_2, \ldots, \succ_m$ sont définies mais une relation de précédence globale doit être trouvée. Cette relation globale (notée $\succ_\mathcal{Q}$) portant sur tout le schéma de la table relationnelle interrogée, doit être telle que toutes les relations $s_1, \ldots, s_m$ sont vérifiées. L'obtention de $\succ_\mathcal{Q}$ par combinaison des relations $\succ_i$ est liée à la sémantique accordée aux expressions de préférences (section 3.1.1).

3.1.2.2 Préférences quantitatives

Dans l'approche quantitative, les préférences sont spécifiées de manière indirecte par des fonctions de score. Un score est calculé pour chaque enregistrement par rapport à la requête et les enregistrements ayant les scores les plus élevés sont préférés aux autres.

Dans les systèmes quantitatifs (par exemple, [83]), l'ordre de préférence est spécifié en attribuant des poids aux critères de sélection, l'importance relative des critères étant fixée par la valeur de leur poids. Des critères d'importances équivalentes sont donc affectés du même poids. Pour une relation $R(A_1, A_2, \ldots, A_n)$, une requête utilisateur précise les poids w_1, w_2, ..., w_n affectés aux critères des attributs $A_1, A_2, \ldots, A_n$. La fonction de préférence suivant laquelle les enregistrements sont triés est donnée par :

$$f_v(t) = w_1 \cdot D_1(t) + w_2 \cdot D_2(t) + \ldots + w_n \cdot D_n(t) \ ,$$

où $D_i(t)$ représente une mesure de distance (ou de similarité) de $t.A_i$ par rapport à la valeur de référence indiquée dans la requête.

Cependant, trouver les meilleurs objets requiert d'examiner tous les enregistrements : la base de données entière est parcourue, ce qui limite les performances en temps de réponse et en capacité de traitement. Pour passer outre cet inconvénient, Hristidis *et al.* [83] proposent la *pré-matérialisation* dans le système PREFER : des requêtes sont prédéfinies et leurs résultats sont stockés. Ces requêtes prédéfinies, ou *vues*, servent ensuite à déterminer l'ensemble des enregistrements qui seront effectivement pris en compte dans le calcul des scores. Lors de l'évaluation d'une requête, le système détermine la vue appropriée pour répondre à la requête en comparant la fonction de préférence de la vue et celle spécifiée dans la requête.

La pré-matérialisation de requêtes soulève des questions qui sont traitées par les auteurs, notamment comment utiliser une vue, comment déterminer un ensemble de vues qui couvrent au mieux l'espace des requêtes possibles ou comment prendre en compte d'éventuelles limitations en espace de stockage.

3.1.3 Problèmes

La principale difficulté dans l'utilisation des préférences tient à leur expressivité, en particulier lorsqu'elles sont quantitatives. Les fonctions de score sont en effet difficiles à construire et à composer. De plus, il n'existe pas toujours une fonction qui retranscrive fidèlement une préférence exprimée en langage naturel. Chomicki montre par exemple dans [34] qu'une fonction de score ne peut retranscrire la préférence : « pour un même livre, je préfère les exemplaires dont le prix est le plus faible ».

De manière plus générale, une fonction de score est inadaptée dès lors que les préférences souhaitées n'induisent pas un ordre total sur tous les objets, c'est-à-dire lorsqu'il existe des objets incomparables. En effet, un tel cas engendre des contraintes qui ne peuvent être satisfaites

par une fonction de score. Par exemple, le moins cher des guides routiers ne peut pas être considéré comme préférable au dictionnaire le moins cher si celui-ci a un prix supérieur. Chomicki propose dans [35] un cadre formel d'expression des préférences par des formules logiques du premier ordre. Ce cadre formel permet de formuler plus ou moins simplement des préférences, quel que soit leur type.

Une autre difficulté, d'ordre pratique, relève de l'efficacité algorithmique du traitement des préférences. Par exemple, la sélection des meilleurs objets en préférences quantitatives requiert (dans le cas général) le calcul d'un score pour tous les enregistrements. Brafman et Domshlak [31] admettent également que la sémantique *ceteris paribus* (voir section 3.1.1) est prohibitive ; ils en proposent une relaxation moins coûteuse mais également moins efficace.

Les problèmes évoqués dans la littérature sur les préférences en bases de données montrent une dualité expressivité / efficacité. D'une part, au vu des comparaisons et autres arguments donnés par Hristidis *et al.* [83], on peut considérer le système PREFER comme une solution au problème d'efficacité lié à l'évaluation des requêtes à préférence dans les bases de données. Mais le problème de l'expressivité de certains types de préférences n'est toujours pas résolu. D'autre part, le cadre (qui résout le problème d'expressivité) proposé par Chomicki [35] où les formules de préférence s'intègrent « de manière naturelle » dans l'algèbre relationnelle, ne dispense pas de traiter systématiquement chaque enregistrement. Ce cadre est donc moins efficace.

3.1.4 Adéquation des résultats

La manière d'apprécier l'adéquation des résultats à la requête de l'utilisateur dépend fortement de la technique utilisée pour définir les préférences. Lorsqu'il s'agit de préférences quantitatives, déterminées par une fonction de score, l'adéquation des résultats est triviale ; les meilleurs résultats sont ceux dont le score est le plus élevé.

En préférences qualitatives, il n'y a (a priori) pas de classement effectué mais l'adéquation reste évidente. D'une manière générale, les résultats correspondant à la forme clausale la plus restrictive sont *préférés* à ceux qui répondent à une expression logique moins restrictive. Ceci découle du sens même des hiérarchies de préférences évoquées en section 3.1.2.1.

Dans un cas comme dans l'autre, on retrouve la même structuration des résultats en strates. Cependant, dans certains travaux sur les préférences qualitatives [90, 94], on note que les résultats retournés dans une réponse sont les meilleurs au regard des préférences exprimées. Il n'y a

donc qu'une strate, celle des résultats qui ne sont *dominés*[1] par aucun autre enregistrement. Le modèle associé est dénommé « Best Matches Only Model ».

Les modèles de préférences qualitatives dans [35] et [90] permettent de rendre compte de l'expression de préférences par le biais de fonctions de score mais avec une limitation en nombre de résultats, car seule la strate supérieure est retournée. En particulier, le modèle BMO de Kießling retourne une réponse composée d'un nombre faible de résultats (un unique enregistrement ou quelques enregistrements).

3.2 Systèmes d'interrogation flexible de bases de données

Dans cette section, nous introduisons quelques systèmes d'interrogation flexible proposés dans la littérature. La description des spécificités de chaque système est accompagnée de celle des constructions, requêtes et résultats admis.

Les systèmes décrits sont, d'une part, des systèmes fondés sur les sous-ensembles flous et, d'autre part, des systèmes étendant le calcul relationnel à l'expression des préférences. L'aspect flexible des derniers tient à cette orientation « prise en compte explicite des préférences ». Quant aux premiers, l'aspect flexible tient essentiellement à la gradualité offerte par les sous-ensembles flous. La gradualité permet d'augmenter les possibilités d'expression en allant au-delà de la bivalence du système binaire. Cependant, ces systèmes d'interrogation qui utilisent des sous-ensembles flous se réclament également du domaine des préférences.

Rappelons qu'un sous-ensemble flou S, défini sur un domaine U, est muni d'une fonction d'appartenance f_S. Un élément x de U est caractérisé par son degré d'appartenance, noté $f_S(x)$ à S. Dubois et Prade [52] ont déterminé trois interprétations de la sémantique d'un sous-ensemble flou : similarité, préférence et incertitude. Dans le cas d'une sémantique de préférence, le sous-ensemble flou S définit explicitement un ordre sur les éléments de U. Il suffit de trier les éléments du domaine en fonction de leur degré d'appartenance au sous-ensemble considéré.

La figure 3.1 montre le sous-ensemble flou décrit par une étiquette « grand », sur le domaine de l'attribut `épaisseur` (déjà présenté en section 1.2). Le noyau du sous-ensemble, c'est-à-dire l'ensemble des valeurs dont le degré d'appartenance à « grand » est maximal est constitué de l'intervalle $[35, 50]$. Par définition, toute valeur issue cet intervalle a une meilleure « adéquation » vis-à-vis de l'ensemble « grand » que toute valeur issue du reste du support, c'est-à-dire

[1] Dans un contexte de requête multi-critères, un enregistrement t_1 est « dominé » par un enregistrement t_2 s'il existe au moins un critère A tel que $t_2.A$ est meilleur que $t_1.A$, les deux enregistrements étant équivalents pour tous les autres critères.

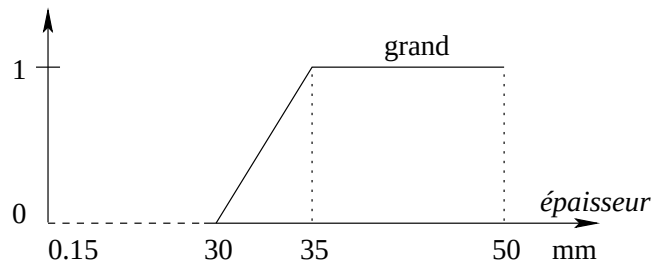


Figure 3.1 – Sous-ensemble flou « grand »

une valeur de $]30, 35[$. Les systèmes d'interrogation basés sur les sous-ensembles flous, forts de la sémantique de préférence attachée aux sous-ensembles flous, interprètent un sous-ensemble flou comme une expression de préférence. Ainsi, la définition de « grand » exprime toutes les précédences $a \succ b$ avec $a \in [35, 50]$ et $b \in]30, 35[$.

3.2.1 SQLf

Le langage d'interrogation SQLf, proposé par Bosc et Pivert [25] est une extension du langage SQL visant à « introduire dans SQL des prédicats flous autant que possible » [24]. L'extension, aussi bien sémantique que syntaxique, permet d'exprimer divers éléments d'une requête sous une forme floue. On trouve parmi ces éléments des opérateurs, des fonctions d'agrégation, des modificateurs linguistiques et des quantificateurs, de même que les variables linguistiques et autres prédicats graduels (voir par exemple la figure 3.2), dont on suppose qu'ils sont « définis au moyen d'une interface appropriée » [22].

En raison du statut d'extension de ce langage, toutes les requêtes SQL y sont valides. Le langage permet en outre de limiter les résultats quantitativement ou qualitativement, et d'intégrer des conditions booléennes et graduelles. La forme générale d'une requête est :

```
SELECT [n | β | n, β] <liste-attr>
FROM <relations>
WHERE <conditions-floues>;
```

où n est un seuil quantitatif et β un seuil qualitatif (c'est-à-dire le degré d'appartenance minimal d'un élément à la réponse).

SQLf détermine le degré d'appartenance d'un enregistrement à la réponse grâce à une algèbre relationnelle étendue aux constructions floues, généralisation de l'algèbre relationnelle. Cette algèbre étendue opère sur des relations graduelles et fournit des relations graduelles en résultat. Les opérations de l'algèbre relationnelle instanciée par SQL sont alors des cas particuliers

des opérations *étendues*. Il s'agit de la sélection, de la projection, de la jointure, d'opérateurs ensemblistes, et de prédicats graduels (par exemple, les prédicats *bien-payé* et *jeune* de la figure 3.2, définis respectivement sur les attributs *salaire* et *âge*).

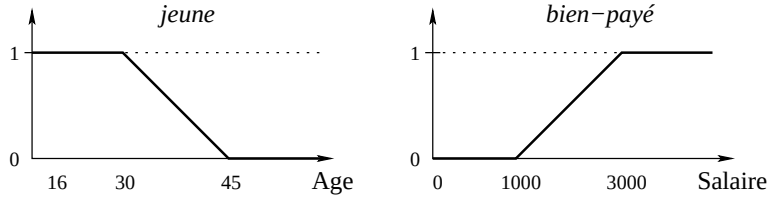


Figure 3.2 – Exemples de prédicats graduels

Une relation R est considérée comme un sous-ensemble flou (de l'espace d'enregistrements représenté) et munie d'une fonction d'appartenance f_R à la relation R. Contrairement à un prédicat flou pour lequel la fonction d'appartenance s'applique aux valeurs d'un domaine d'attribut (voir figure 3.2), f_R s'applique aux enregistrements de la relation. Dans le cas d'une relation classique (telle que définie par Codd [40]), la fonction d'appartenance est booléenne et invariante : $\forall\, t \in R, f_R(t) = 1$ et $\forall\, t \notin R, f_R(t) = 0$. Pour une relation graduelle, si la relation R est une table de la base de données, la fonction est également invariante, mais si la relation résulte de l'application d'opérateurs étendus, la fonction devient graduelle. Par exemple, la relation R de la table « Employés » (tableau 3.1) voit ses enregistrements affectés de degrés, résultant de l'application des prédicats *bien-payé* et *jeune* (respectivement, tableau 3.2 et tableau 3.3), pour définir les relations $\mathrm{R}_{bien-payé}$ des employés bien payés et R_{jeune} des employés jeunes.

Table 3.1 – Extrait de la relation R

Numéro	Nom	Département	Salaire	Âge	...
1	Alice	RH	5.600	62	
2	Jonathan	CPT	1.500	33	
3	Michael	TECH	2.200	42	
4	Suzanne	RD	3.200	25	
5	Paul	RH	4.000	50	
...	...	...	...	...	

Notons que plusieurs interprétations sont parfois disponibles. C'est ainsi que les degrés résultant de *jeune ET bien-payé* ne sont pas les mêmes suivant l'interprétation de l'opérateur

Table 3.2 – Relation graduelle $R_{bien-payé}$

Numéro	Nom	Département	Salaire	Âge	...	$\alpha_{bien-payé}$
1	Alice	RH	5.600	62		1,0
2	Jonathan	CPT	1.500	33		0,25
3	Michael	TECH	2.200	42		0,6
4	Suzanne	RD	3.200	25		1,0
5	Paul	RH	4.000	50		1,0

Table 3.3 – Relation graduelle R_{jeune}

Numéro	Nom	Département	Salaire	Âge	...	α_{jeune}
1	Alice	RH	5.600	62		0,0
2	Jonathan	CPT	1.500	33		0,8
3	Michael	TECH	2.200	42		0,2
4	Suzanne	RD	3.200	25		1,0
5	Paul	RH	4.000	50		0,0

flou matérialisant *ET*. Pour plus de détails sur les possibilités d'interprétation, le lecteur est invité à consulter [22].

On note également le cas particulier de l'opérateur d'égalité, dont seule la sémantique est modifiée. Ainsi, l'égalité, dénotée syntaxiquement par « = », n'est plus une égalité stricte (booléenne), mais est traduite par le degré d'appartenance au sous-ensemble flou indiqué. On peut ainsi écrire :

```
SELECT 3 Nom FROM EMPLOYES WHERE Age = 'jeune';
```

SQLf étend également les connecteurs inter-requêtes (IN, NOT IN, EXISTS) de manière à ce que l'équivalence sémantique éventuelle entre 2 requêtes SQL syntaxiquement différentes reste respectée dans SQLf.

3.2.2 FQUERY

FQUERY [87] est un effort d'application du paradigme « *computing with words* » [153] (encore désigné par « *soft querying* ») aux bases de données. Cette proposition vise une ges-

tion efficace de « formalismes capables de traiter l'ambigüité, la gradualité et l'imprécision du langage naturel ». Pour les auteurs, la mise en pratique de ce paradigme passe non par la construction de SGBD flous, mais par des modules de gestion du flou qui se grefferaient aux SGBD classiques, largement disponibles et éprouvés. Les sous-ensembles flous et la logique floue servent de bases théoriques à la proposition.

FQUERY permet de spécifier des valeurs linguistiques sur les attributs (*élevé, moyen, ...*), des relations linguistiques (*proche de, supérieur à, ...*), des modificateurs linguistiques (*très, plus ou moins, ...*) et des quantificateurs linguistiques (*la plupart, peu, ...*). L'introduction de ces éléments dans la requête conserve la nature des résultats : la réponse est toujours constituée d'enregistrements de la base de données. Les requêtes que l'on peut formuler sont sous la forme :

trouver les enregistrements tels que *la plupart* de leurs valeurs sur les attributs *importants* répondent aux *descriptions* spécifiées

Comme le montre cette forme générale, il est également possible d'associer des niveaux d'importance différents aux attributs, assimilables à des pondérations. FQUERY requiert la définition préalable des sous-ensembles flous correspondant aux éléments linguistiques. La requête résultante est également considérée comme un sous-ensemble flou, obtenu par application d'opérateurs de la logique floue définis par Yager [149]. Par conséquent, chaque élément de l'ensemble (c'est-à-dire chaque enregistrement t résultat) appartient au sous-ensemble flou de la requête à un certain degré α_t. Les résultats fournis en réponse à la requête sont triés suivant la valeur des degrés d'appartenance. Ils font apparaître explicitement le degré d'appartenance de chaque enregistrement.

3.2.3 FSQL

FSQL [70] est également une extension syntaxique et sémantique du langage SQL. Le langage est implémenté dans une architecture client-serveur où le serveur gère l'accès à la base de données. Le langage SQL est étendu de manière à permettre des conditions flexibles utilisant des constructions floues (étiquettes linguistiques, constantes, etc.).

Les éléments suivants se retrouvent dans le langage FSQL :

- une base de métadonnées floues (*Fuzzy Metaknowledge Base*) où sont définis les étiquettes et quantificateurs linguistiques, permettant une réutilisation plus facile. Une métadonnée est spécifiée par quatre valeurs définissant un sous-ensemble flou trapézoïdal ;

- des opérateurs de comparaison flous, extensions d'opérateurs classiques ($=$, $\neq$, $<$, $\leq$, ...), déclinés en deux versions dotées respectivement d'une sémantique de possibilité ou d'une sémantique de nécessité ;
- des constantes floues : il s'agit de valeurs particulières (`UNKNOWN, UNDEFINED, NULL`), de valeurs approximatives, d'intervalles, ou de distributions de possibilité ;
- des seuils de satisfaction : attachés à une condition floue, ils déterminent le degré de satisfaction minimal pour prendre en considération l'élément évalué (typiquement, un enregistrement) ;
- des opérateurs ensemblistes flous.

FSQL se distingue par la grande variété de constructions disponibles et par son orientation pratique, mais aussi par l'extension d'autres clauses que la clause SELECT, traditionnellement visée dans les autres travaux. La syntaxe des clauses INSERT, DELETE et UPDATE est ainsi capable d'accepter des expressions, conditions et requêtes floues. Signalons qu'une implémentation de FSQL a été réalisée et est disponible pour le SGBD Oracle.

3.2.4 PREFERENCES

Le système PREFERENCES de Lacroix et Lavency [94] est une extension aux préférences applicable à tout langage de la famille *Domain Relational Calculus* (DRC). Le DRC, introduit par E. F. Codd en 1972 [41], est une généralisation en logique du premier ordre des langages déclaratifs d'interrogation de bases de données. Le langage descriptif utilisé dans les exemples ci-dessous est équivalent à toute autre instance du DRC (par exemple, SQL).

La motivation première de PREFERENCES a été « la difficulté d'exprimer, dans les langages traditionnels, les caractéristiques désirables des objets à sélectionner ». Ce travail ayant été l'un des premiers à traiter des préférences, la plupart des généralités exposées dans les sections précédentes, notamment à propos des préférences qualitatives (section 3.1.2.1), s'y appliquent.

Le contexte des travaux est celui de l'ingénierie logicielle mais la proposition est généralisable sans restriction. Le langage admet des clauses de préférences simples, imbriquées (ou hiérarchiques), ou de même niveau (préférences cumulatives). Le tableau 3.4 donne la syntaxe de ces trois types de clauses. Nous reprenons l'exemple considéré dans [94], celui d'une base de données des codes sources multiversions d'une application composée de plusieurs modules.

On pourrait considérer que les enregistrements résultats font partie de différentes strates indicatives de leur niveau de satisfaction vis-à-vis des préférences. La relation de précédence de la section 3.1.2.1 permet alors de distinguer ces strates. Mais PREFERENCES limite le nombre

Table 3.4 – Clauses de préférences dans PREFERENCES

Préférence simple	Préférences imbriquées	Préférences cumulatives

`from which`
` prefer those`
`  having STATUS=coded`
	`from which`
	` prefer those`
	`  having STATUS=coded`
	`from which`
	` prefer those`
	`  having AUTHOR=Pierre`
		`from which`
		` prefer those`
		`  having STATUS=coded`
		` prefer those`
		`  having AUTHOR=Pierre`

de strates à 1 : la réponse est composée des objets satisfaisant le maximum de préférences. On ne retrouve donc pas de résultats satisfaisant des nombres différents de clauses de préférences. Par exemple, dans le cas des préférences cumulatives du tableau 3.4, les versions satisfaisant la condition `STATUS=coded` mais dont Pierre n'est pas l'auteur n'apparaîtront qu'en l'absence de versions satisfaisant la condition et réalisées par Pierre.

3.2.5 Preference SQL

Preference SQL [91] est l'application à SQL d'un modèle d'expression de préférences dans les langages d'interrogation de bases de données [90]. Le modèle définit une préférence simple comme un ordre partiel strict (entre enregistrements d'une relation) sur la base de la valeur d'un attribut. Soit une préférence P exprimée sur un attribut A de domaine D, la relation $\succ$ associée à P est irréflexive, asymétrique et transitive :

- $\forall\, x \in D, \neg(x \succ x)$
- $\forall\, x \in D, \forall\, y \in D, \quad (x \succ y) \Rightarrow \neg(y \succ x)$
- $\forall\, (x, y, z) \in D^3, \quad (x \succ y) \wedge (y \succ z) \Rightarrow x \succ z$

Une relation $\prec$, réciproque de la relation $\succ$, définit une préférence P' car elle est également une relation d'ordre partiel strict. Les deux relations peuvent être représentées par un même graphe orienté acyclique, la distinction se faisant sur l'orientation des arcs (figure 3.3). Les valeurs de l'attribut dans le graphe sont disposées par niveaux reflétant bien l'ordre partiel à l'origine du graphe (par exemple, le jaune et le blanc sont incomparables). Dans le modèle, P' est considéré comme la *préférence duale* de P (notons que le dual mathématiquement correct de $\succ$ est un ordre large – c'est-à-dire réflexif – plutôt qu'un ordre strict tel que $\prec$).

À partir de cette définition, une préférence élémentaire peut être formulée. Sur un attribut non numérique, elle peut indiquer des caractéristiques souhaitées, dites *positives*, spécifiant

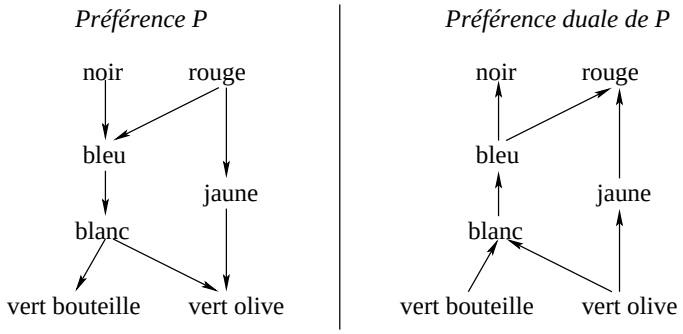


Figure 3.3 – Graphes de préférences

ainsi le niveau supérieur du graphe. Elle peut indiquer des caractéristiques rejetées, dites *négatives*, équivalentes au niveau inférieur du graphe. Par exemple, si nous reprenons le scénario des véhicules d'occasion (section 3.1.1), la préférence positive `POS(couleur, {rouge, noir})` indique une précédence des couleurs rouge et noire sur les autres couleurs. La préférence négative `NEG(couleur, {bleu})` place la couleur bleue au niveau inférieur du graphe.

Les préférences positives et négatives peuvent être répétées ou apparaître conjointement. C'est ainsi que

```
POS/POS(couleur, {rouge, noir}, {bleu})
```

exprime la précédence des couleurs rouge et noire sur la couleur bleue, elle-même précédant les autres couleurs. De même,

```
POS/NEG(couleur, {rouge, noir}, {bleu})
```

place le rouge et le noir au niveau supérieur du graphe de préférences, et le bleu au niveau inférieur.

Pour un attribut numérique, les constructions précédentes restent valables. En outre, une fonction de distance peut se substituer au niveau dans le graphe. On se ramène alors à des préférences quantitatives (section 3.1.2.2) sur l'attribut concerné. Une préférence exprime dans ce contexte que la distance est considérée comme optimale si elle est minimale par rapport à :

- une valeur spécifiée : `AROUND (nombre de places, 5)` ;
- un intervalle spécifié : `BETWEEN (kilométrage, [15.000, 50.000])` ;
- la borne (inférieure ou supérieure) du domaine : `LOWEST(prix)` ;
- le résultat d'une fonction de score spécifiée : `SCORE(puissance, `f`)`.

Il est possible d'obtenir un large éventail de préférences complexes par des opérateurs de composition de préférences. Quelques opérateurs sont présentés ci-après pour P_1 et P_2, de relation respective $\succ_1$ et $\succ_2$, définies sur les attributs A_1 et A_2 :

- l'accumulation (*pareto preference*) ($P_1 \otimes P_2$) :

$$(x_1, x_2) \succ (y_1, y_2) \iff [x_1 \succ_1 y_1 \wedge (x_2 \succ_2 y_2 \vee x_2 = y_2)] \vee [x_2 \succ_2 y_2 \wedge (x_1 \succ_1 y_1 \vee x_1 = y_1)]$$

- la hiérarchisation (*prioritized preference*) ($P_1 \,\&\, P_2$) :

$$(x_1, x_2) \succ (y_1, y_2) \iff [x_1 \succ_1 y_1] \vee [x_2 \succ_2 y_2 \wedge (x_1 = y_1)]$$

- l'intersection ($P_1 \blacklozenge P_2$) :

$$(x_1, x_2) \succ (y_1, y_2) \iff [x_1 \succ_1 y_1] \wedge [x_2 \succ_2 y_2]$$

- l'union disjointe ($P_1 + P_2$), où $A_1 = A_2$:

$$(x_1, x_2) \succ (y_1, y_2) \iff [x_1 \succ_1 y_1] \vee [x_2 \succ_2 y_2]$$

Le modèle offre également une algèbre des préférences qui permet d'exprimer les préférences en logique du premier ordre. Les requêtes à préférence, après transformation, se présentent sous la forme de requêtes SQL auxquelles la clause `PREFERRING` rajoute des préférences. On obtient ainsi les exemples de requêtes du tableau 3.5.

Les résultats obtenus sont présentés comme des résultats d'une interrogation classique avec SQL. La différence tient au contenu de la réponse, en termes d'adéquation avec les préférences de l'utilisateur.

3.3 Conclusion

Dans ce chapitre, l'interrogation flexible de bases de données a été présentée. Partant de la manière dont l'interrogation dite *flexible* est évoquée dans la littérature, nous avons tenté de combler l'absence d'une définition en en proposant une. Ainsi, l'interrogation flexible de bases de données couvre toute interrogation où la séquence [expression de la requête → évaluation → présentation des résultats] ne respecte pas strictement le schéma de l'interrogation *classique* à base du langage SQL, dans le but d'enrichir l'interrogation. Entrent dans cette catégorie les systèmes qui :

Table 3.5 – Exemples de requêtes dans Preference SQL

Préférences de base	`SELECT * FROM Vehicules` `PREFERRING couleur IN ('rouge', 'noir');`
	`SELECT * FROM Vehicules` `PREFERRING LOWEST(prix);`

Accumulation	`SELECT * FROM Vehicules`
	`PREFERRING couleur IN ('rouge', 'noir')`
	`AND LOWEST(prix);`

Hiérarchisation	`SELECT * FROM Vehicules`
	`PREFERRING marque IN ('Mini', 'Smart')`
	`CASCADE couleur IN ('rouge', 'noir')`
	`CASCADE LOWEST(prix);`

- prennent en charge des données non fortement structurées ;
- facilitent l'expression des besoins en termes de requêtes ;
- présentent un aspect coopératif ;
- fournissent des réponses approximatives.

Nous avons mis en évidence la part importante des *préférences* dans la perception actuelle de l'interrogation flexible. Une préférence est un souhait exprimé par l'utilisateur émettant une requête. Ce souhait peut être considéré comme un critère *souple* de la requête : c'est une indication supplémentaire pour le système chargé d'y répondre. À l'évaluation de la requête, on fait l'hypothèse implicite qu'un objet sélectionné à l'issue de l'évaluation offrira une satisfaction plus grande à l'utilisateur s'il entre dans le cadre du souhait exprimé. L'évaluation d'une requête favorise donc les enregistrements les plus satisfaisants en regard des préférences. Ces enregistrements sont ensuite mis en avant à la présentation des résultats, en l'occurrence, placés en tête de liste.

Ce chapitre aborde également des généralités sur les préférences. Ainsi, on distingue les préférences explicites, où les expressions de préférence font partie du langage de requête, des préférences implicites où les souhaits se traduisent par des poids ou des scores. En outre, le problème de la sémantique d'une préférence exprimée sur un attribut lors de la prise en compte des autres attributs est évoqué. On montre que l'interprétation la plus correcte (suivant l'intuition humaine) n'est pas celle adoptée dans le traitement des préférences en raison d'une complexité plus importante et d'un pouvoir discriminant plus faible que l'interprétation utilisée en pratique.

CHAPITRE 4

Application des résumés SAINTETIQ à l'interrogation flexible

Introduction

Ce chapitre traite de l'interrogation des résumés linguistiques de données produits par le modèle SAINTETIQ dans le cadre de l'interrogation flexible. La réponse usuelle à une requête de bases de données, désignée par le terme « réponse précise », est constituée d'enregistrements de la relation interrogée. Les valeurs des enregistrements résultats sont alors précisément connues. L'interrogation d'une hiérarchie de résumés, vue au chapitre 2, fournit une réponse dite « approchée » car moins détaillée qu'une réponse précise visant les mêmes données. En effet, les résumés résultats fournissent une description des données, plus générale que les valeurs précises des enregistrements. La nature de la requête, précise ou flexible (lorsqu'elle est exprimée par des termes linguistiques), et la nature de la réponse, précise ou approchée, permettent de distinguer quatre cas d'interrogation où les résumés SAINTETIQ peuvent intervenir.

Pour une réponse approchée, l'algorithme 1 du chapitre 2 sera repris tel quel. On considère que les critères de sélection sont fournis par une requête flexible, c'est-à-dire une requête dont les critères sont spécifiés par des termes linguistiques (par exemple, `âge IN jeune`). On peut également envisager le cas d'une requête précise, dont les valeurs de critères sont issus des domaines d'attribut (par exemple, `âge = 25`). La réponse restant approchée dans ces deux cas, le processus de recherche est le même. Cependant, une étape préalable de traduction de la valeur précise (`25`) par une caractérisation linguistique (*jeune*) permet de mettre les critères de sélection dans le format adéquat. Le mécanisme régissant cette traduction est déjà utilisé dans la réécriture des enregistrements en tuples candidats (voir section 1.3.1). Il ne présente pas de difficulté particulière et est réutilisable tel quel pour ce cas (requête précise – réponse approchée).

Pour une réponse précise, c'est-à-dire composée des enregistrements satisfaisant les conditions de sélection, le traitement effectué s'apparente à une interrogation dans une base de données relationnelle par le biais du langage SQL. Ici, les résumés jouent le rôle d'un index. En raison de la complexité des index, nous reportons la discussion de cet aspect de l'utilisation des résumés aux chapitres suivants. Le dernier cas d'interrogation est celui où une réponse précise est fournie à partir d'une requête flexible. Ce cas est une extension de la réponse approchée : les enregistrements décrits par les résumés résultats sont rendus disponibles. Cette fonctionnalité, qui revient à présenter les résultats avec un niveau de détail supérieur, ne requiert qu'une connexion avec la base de données pour être réalisable. Elle ne sera donc pas abordée ci-après.

Dans la suite du chapitre, l'implémentation de l'algorithme d'interrogation est présentée, accompagnée d'une évaluation du processus en termes de temps d'exécution. Des enrichissements du processus sont également décrits, notamment la modification de requêtes, qui cherche à apporter une réponse aux requêtes sans résultat, et l'utilisation dans la requête d'un vocabulaire différent de celui de la construction des résumés.

4.1 Interrogation approchée des données

4.1.1 Implémentation

L'algorithme d'interrogation des résumés du modèle SAINTETIQ, présenté au chapitre 2 a été implémenté dans un prototype d'interrogation approchée des données qui reprend tous les principes qui y ont été présentés :

- la requête est spécifiée en donnant les descripteurs linguistiques requis ;
- l'accès à la base de données n'est pas utilisé ;
- l'interrogation réalise une description des données satisfaisant les descripteurs requis ;
- les résultats de l'interrogation sont des résumés ;
- les résultats sont présentés par *classes* (voir section 2.2.4) sous une forme plus intelligible qu'une liste de résumés.

Les requêtes sont exprimées à travers une interface qui permet à l'utilisateur de composer la requête sans connaissance du pseudo-langage formel utilisé. Les attributs requis sont d'abord choisis (voir figure 4.1). Pour chacun, les caractérisations recherchées sont ensuite indiquées. Étant donné que les interprétations sont implicites, respectivement conjonctive (inter-attributs) et disjonctive (sur un même attribut), il n'existe pas de choix d'opérateur.

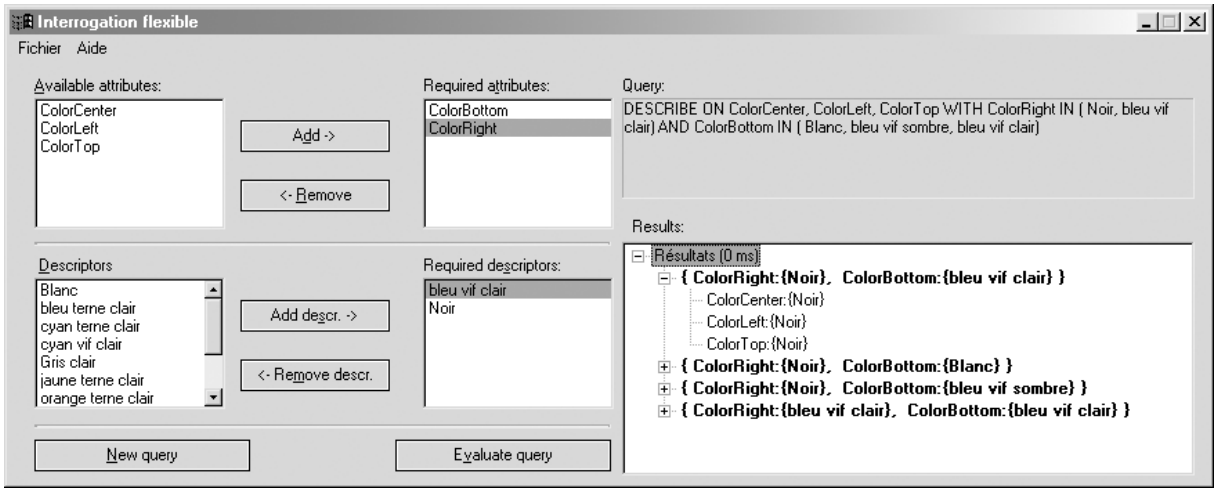


Figure 4.1 – Implémentation de l'interrogation de résumés

Les résultats présentés par l'outil couvrent les deux applications évoquées dans le chapitre 2 : le test d'existence de données répondant à un ensemble de caractérisations, et la description de ces données. Dans ce dernier cas, la notion de *classe* est matérialisée par la distinction entre les différentes combinaisons d'attributs requis trouvées au sein des données. Par exemple, la figure 4.1 montre un exemple de requête et de résultats sur un jeu de données portant sur une base d'images. Les images sont partitionnées en cinq zones (*Top*, *Bottom*, *Left*, *Right*, *Center*) suivant le schéma de la figure 4.2. Elles sont décrites par les couleurs dominantes de chaque zone (respectivement *ColorTop*, *ColorBottom*, *ColorLeft*, *ColorRight* et *ColorCenter*). La requête formulée dans cet exemple est :

```
DESCRIBE ON ColorCenter, ColorLeft, ColorTop WHERE
  ColorRight IN {Noir, bleu vif clair} AND
  ColorBottom IN {Blanc, bleu vif sombre, bleu vif clair}  .
```

La réponse à cette requête fournit des résultats synthétisés dans quatre classes (texte en gras, en bas à droite, dans la figure 4.1) :

1. {ColoRight : {Noir}, ColorBottom : {bleu vif clair}} ;
2. {ColoRight : {Noir}, ColorBottom : {Blanc}} ;
3. {ColoRight : {Noir}, ColorBottom : {bleu vif sombre}} ;
4. {ColoRight : {bleu vif clair}, ColorBottom : {bleu vif clair}}.

Les avantages d'une présentation des résultats par classes sont également visibles. Ainsi, on sait que dans le jeu de données utilisé, des données décrites par *bleu vif clair* sur ColorRight et *Blanc*

sur ColorBottom n'existent pas, ou que les images de la première classe ({ColoRight : {Noir}, ColorBottom : {bleu vif clair}}) dont le noir est la couleur dominante à droite, présentent la même couleur au centre, à gauche et sur leur partie supérieure.

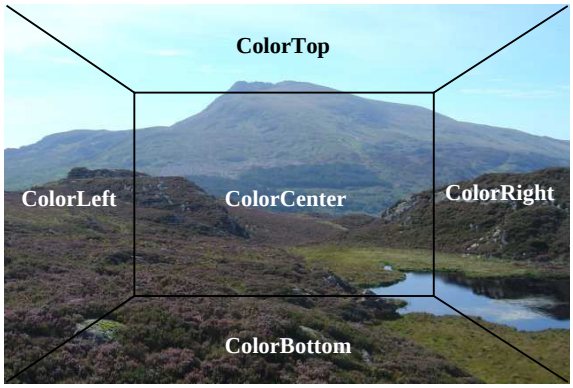


Figure 4.2 – Partitionnement d'une image en cinq zones

Cette implémentation peut être enrichie de diverses informations lors de la présentation des résultats. Il s'agit notamment des résumés sélectionnés, des degrés et mesures du modèle (par exemple, le nombre de tuples couverts par chaque classe), ou de valeurs déterminées par des besoins spécifiques (calculs d'agrégats, etc.). Il est également possible d'associer aux résultats les enregistrements participant aux résumés sélectionnés. Cette option permettra de visualiser les valeurs d'attributs, offrant ainsi un niveau de détail supplémentaire à l'interrogation des résumés.

4.1.2 Expérimentation

Cette section décrit une évaluation de l'algorithme d'interrogation des résumés. L'objectif est de mesurer les temps de réponse pour une variété de requêtes. Pour ce faire, deux jeux de données, dont le tableau 4.1 présente les caractéristiques, ont été utilisés. Le tableau montre le nombre d'attributs (colonne « Dimension ») et le nombre d'enregistrements (colonne « Tuples ») de la relation, le nombre de termes linguistiques dans les connaissances de domaine, et des caractéristiques de la hiérarchie construite. Chaque jeu de données correspond à une relation, portant sur des images ou des données bancaires du groupe CIO, dont la hiérarchie de résumés, stockée dans un fichier XML, est utilisée en entrée de l'outil d'interrogation. Le jeu de requêtes utilisé pour évaluer l'algorithme d'interrogation est détaillé ci-après.

Table 4.1 – Jeux de données

	Relation			Hiérarchie de résumés		
Nom	Dimension	Tuples	Termes	Nœuds	Feuilles	Profondeur
IMAGES	5	169	56	131	74	10
CIO	10	33.733	34	27.304	14.269	23

On suppose qu'une position binaire est affectée à un terme linguistique du vocabulaire des connaissances de domaine (tableau 4.2). Ne sont pris en compte que les termes **effectivement présents** dans la hiérarchie. Ceci permet d'exclure les requêtes dont on est sûr qu'elles n'auront pas de résultat : cette certitude est acquise dès la racine de l'arbre. Aucune exploration ne survient.

Sur la base de l'hypothèse d'affectation des positions binaires, un parcours exhaustif de l'espace des requêtes peut être effectué en simulant une énumération de valeurs entières. Un bit à 1 indique que le descripteur affecté à la position est un caractère requis pour la requête à évaluer. L'énumération des valeurs démarre à 1, dont la représentation binaire est :

$$00\ldots0001\ .$$

La première requête sur la relation IMAGES est ainsi $\mathcal{Q}_1$, dont la condition de sélection est :

```
ColorTop IN (bleu vif clair) .
```

Le descripteur *bleu vif clair* est requis car il est affecté au bit de poids faible, qui est posé lorsque l'énumération est à 1. La valeur suivante dans l'énumération est 2 ($00\ldots0010$). Cette valeur simule une deuxième requête, $\mathcal{Q}_2$ avec la condition `ColorTop IN (noir)` car *noir* correspond au bit 2. La troisième requête est simulée par la valeur 3 ($00\ldots0011$), qui active deux positions binaires. Les descripteurs affectés à ces positions apparaissent donc dans le critère de sélection de $\mathcal{Q}_3$: `ColorTop IN (bleu vif clair, noir)`.

En continuant l'énumération par incrémentation, toutes les requêtes qu'il est possible de former en utilisant les descripteurs des connaissances de domaine, peuvent être obtenues. Mais le parcours de tout l'espace des requêtes, rapporté au nombre de positions (56 et 34 respectivement pour IMAGES et CIO), et à l'exécution de la requête entre deux valeurs consécutives, est une opération très longue. Pour réduire le temps d'attente, et puisque l'expérience est destinée à évaluer les temps de réponse de l'algorithme d'interrogation, l'énumération est arrêtée une fois un certain seuil atteint (en l'occurrence, 2.000). Le nombre de requêtes est donc borné.

Table 4.2 – Affectation de positions

Attribut	Position	Descripteur
`ColorTop`	1	bleu vif clair
	2	noir
	3	bleu terne clair
	4	rouge terne clair
	5	orange vif clair
	6	gris clair
	7	vert terne clair
	8	blanc
	9	orange terne clair
	10	rouge vif clair
	11	jaune terne clair
`ColorBottom`	12	gris clair
	13	noir
	14	orange terne clair
	...	...
	...	...
`ColorCenter`	46	noir
	...	...
	56	rouge terne clair

L'environnement matériel et logiciel de l'expérimentation est le suivant :

- Windows 2000 Professionnel, Service Pack 4 ;
- multitâche en temps partagé avec 3 quantums en 10 ms, répartition standard de 9 quantums par processus [103] ;
- résolution du temporisateur : 1 ms ;
- CPU : monoprocesseur Pentium IV @ 1,7 Ghz ;
- RAM : 768 Mo

On notera que la hiérarchie conserve le format XML de stockage sur disque. La quantité de mémoire consommée n'est pas optimale, mais un monitoring de la mémoire montre qu'aucune pagination n'est survenue pendant l'exécution des requêtes.

Les figures 4.3 et 4.4 présentent les résultats obtenus sous la forme d'histogrammes de fréquence : une barre de l'histogramme indique le nombre d'occurrences de la valeur (ou la plage de valeurs) en abscisse parmi les temps d'exécution mesurés. La somme des fréquences est donc égale au nombre de requêtes exécutées (2.000). Dans la figure 4.3, une barre de l'histogramme correspond à une valeur de temps d'exécution (0, 10 et 20 ms). Dans le cas de la relation CIO, la hiérarchie de résumés est plus importante et les valeurs distinctes plus nombreuses. Les valeurs de temps d'exécution sont groupées par pas de 50 ms dans le graphique.

On constate que les temps d'exécution mesurés sont faibles. On obtient une moyenne arithmétique de 0,3455 ms pour la relation IMAGES et de 41,715 ms pour CIO.

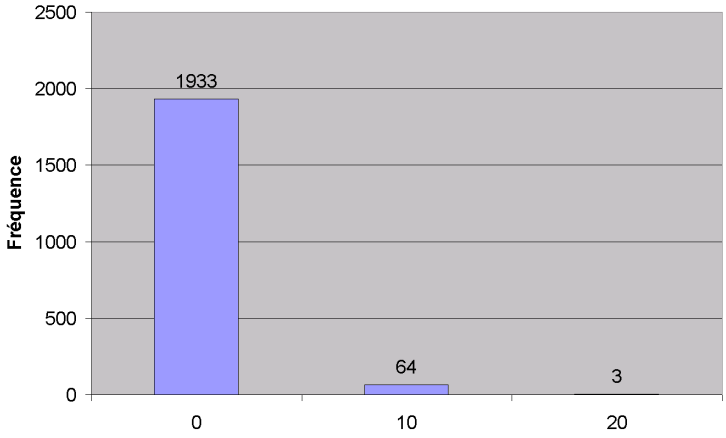


Figure 4.3 – Temps d'exécution sur la relation IMAGES

4.2 Amélioration du processus de construction des résumés

La construction d'une hiérarchie de résumés est un processus incrémental (voir le chapitre 1) où tous les enregistrements de la relation R à résumer suivent les mêmes étapes. Premièrement, les connaissances de domaine sont utilisées pour réécrire un enregistrement t sous la forme de tuples candidats. Ensuite, chaque candidat ct passe par une classification conceptuelle. À cette étape, ct progresse de proche en proche de la racine courante de la hiérarchie (z_0) jusqu'à une feuille z_f. En section 1.3.2, il a été dit que le chemin suivi par ct est découvert de proche en proche par application d'opérateurs d'apprentissage qui déterminent, suivant la description de ct et la hiérarchie courante, le meilleur opérateur à appliquer.

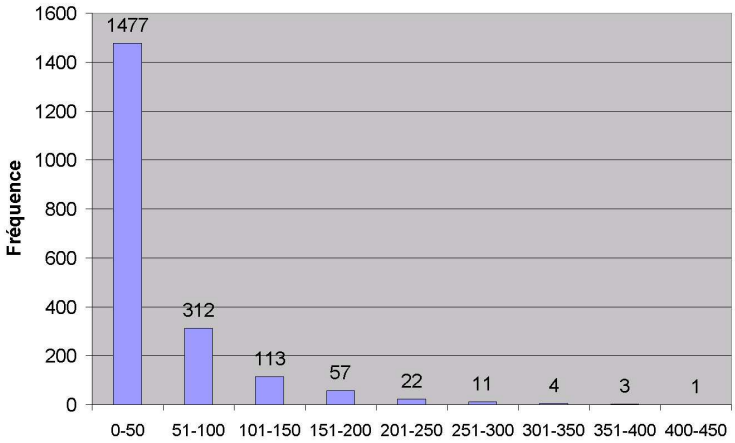


Figure 4.4 – Temps d'exécution sur la relation CIO

L'algorithme d'interrogation des résumés peut être utilisé afin d'améliorer les performances du processus de construction. En effet, ce processus ne tient pas compte du fait que la feuille z_f est entièrement déterminée dès que la réécriture du tuple t est effectuée. Lorsque le tuple candidat ct est obtenu, l'ensemble des termes linguistiques qui entrent dans son expression en intension est identique à l'ensemble des termes de z_f.

Le principe de l'interrogation de résumés comme heuristique est le suivant : le nœud z_f étant unique, l'interrogation permet, à partir de ct, de déterminer si z_f existe. Si oui, l'application des opérateurs d'apprentissage pour ct n'est plus utile car le chemin de z_0 à z_f est reconnu par l'interrogation sans calcul de score ni choix consécutif au calcul. Ne reste plus alors qu'à mettre à jour les degrés et mesures sur le chemin menant à la feuille : l'incorporation de ct se résume à une classification car la structure de l'arbre ne change pas.

Dans le cas où z_f n'existe pas encore, les opérateurs peuvent s'appliquer. Le surcoût engendré par la recherche de z_f est largement compensé par le gain lié à la non-application des opérateurs d'apprentissage à chaque nœud depuis la racine, et ce, pour chacun des tuples candidats de la feuille. Ainsi, quel que soit le volume de l'extension d'une feuille, les opérateurs structurels ne s'appliqueront qu'une fois, lors de l'incorporation du premier candidat couvert par la feuille.

Le principe énoncé ci-dessus entre en conflit avec le rôle des opérateurs d'apprentissage. Ce rôle est de structurer la hiérarchie en cours de construction, c'est-à-dire décider des regroupements et du placement des nœuds internes (hors feuilles et racine). Or, le fait d'utiliser les opérateurs uniquement pour le premier candidat incorporé soulève la question de l'*optimalité*

de la hiérarchie alors obtenue. On peut remarquer que le processus de construction n'offre pas de garantie d'optimalité. De plus, les statistiques qu'il livre montrent que la structure de l'arbre est dictée principalement par les premières insertions [127]. D'autre part, les regroupements et le placement des nœuds internes importent peu dans certaines applications, en particulier lorsque l'arbre des résumés agit comme structure d'index. Dans ce dernier cas, la recherche dans l'arbre ne sélectionne que les feuilles, où sont stockées les références vers les enregistrements. La structure de l'arbre sert uniquement à guider la recherche. On en déduit que la non-application des opérateurs d'apprentissage est réservée à des applications spécifiques.

4.3 Modification de requêtes

L'objectif visé dans cette section est d'offrir une réponse en l'absence de résumés correspondant strictement à la requête telle que considérée jusqu'ici. La modification que nous exposons ci-dessous s'appuie sur l'idée qu'il pourrait exister des résultats sémantiquement proches de ceux recherchés par l'utilisateur. Pour trouver ces résultats, la requête est modifiée en utilisant des informations préétablies ou les résumés eux-mêmes : modifier une requête revient à remplacer la requête sans résultat par une ou plusieurs autres requêtes. Ce comportement est similaire à la « réparation de requêtes » déjà implémentée dans le contexte d'un médiateur par Bidault *et al.* [17, 18]. De par son aptitude à faciliter la réécriture de requêtes, il fait partie des comportements coopératifs listés par Gaasterland *et al.* dans [64].

Dans la suite, les aspects coopératifs présents dans les systèmes d'interrogation sont présentés. Puis, l'ajout d'un comportement coopératif à l'interrogation des résumés décrite dans le chapitre 2 sera traité.

4.3.1 Aspects coopératifs

Les travaux sur les réponses coopératives aux requêtes remontent à ceux de Gal [69] et Cuppens et Demolombe [47] en 1988, dont l'orientation vers les bases de données déductives est très affirmée. D'autres travaux consécutifs, par exemple ceux de Gaasterland *et al.* [64, 66, 67] ou de Minker [104], portant sur les réponses coopératives sont également ancrés dans le domaine des BD déductives.

Les réponses coopératives s'inspirent d'autres travaux sur l'information extraite des conversations entre humains. Dans les conversations, les cycles question-réponse font implicitement intervenir des hypothèses, aussi bien lors de la formulation des questions, que lors de l'opération qui consiste à rassembler les informations nécessaires. Ces hypothèses constituent ce que David

Sadek [124] considère comme un protocole conversationnel : il donne l'exemple de la question « auriez-vous l'heure, s'il vous plaît ? » . La réponse « oui » à cette question est une réponse valide et précise. Mais cette réponse est jugée « anormale ». En effet, la personne qui pose la question fait l'hypothèse qu'elle obtiendra l'heure courante. Cette attente, par ailleurs légitime, n'étant pas remplie, la conversation requerra un autre cycle question-réponse pour conduire à une satisfaction acceptable.

Les quatre « principes de coopération » suivants, définis par Herbert Grice [74, 75] dans le cadre des conversations, sont rapportés dans les travaux sur les réponses coopératives (voir par exemple, [15, 69]) :

1. le principe de qualité, qui garantit un résultat valide et accompagné, si besoin est, des réserves nécessaires pour ne pas induire en erreur ;
2. le principe de quantité, particulièrement pertinent vis-à-vis des masses de données, préconise une quantité d'informations et un niveau de détail de la réponse ni trop bas, ni trop élevés ;
3. le principe de style, qui favorise la communication de l'information grâce à un effort visant à éliminer les ambigüités ;
4. le principe de relation, qui vise le maximum de pertinence entre l'intention de « l'interrogateur » et la réponse fournie.

En interrogation de bases de données, le principe de qualité est pleinement satisfait dès que la recherche est complète et exacte : tous les résultats valides sont sélectionnés et aucun enregistrement non-valide ne fait partie de la réponse. Une réponse approchée (au sens de l'approximation discutée en section 1.1.2) peut également satisfaire ce principe de qualité, qui semble être la première contrainte d'un système d'interrogation.

Le principe de quantité est rencontré de façon plus marginale, notamment dans une base de données classique, où le langage SQL requiert explicitement de fixer le nombre de résultats par une clause `LIMIT`, et où le concept de « détail » n'a pas cours. On notera néanmoins que certaines propositions, par exemple CoBase [36] et les méthodes de résumés, prennent en charge plusieurs niveaux d'abstraction grâce à une organisation hiérarchique. De même, les systèmes de préférences (voir chapitre 3) peuvent inclure des clauses limitant la quantité de données ; c'est le cas de PreferenceSQL [91]. En conséquence, le niveau de détail (et donc la quantité de données) peut être adapté à une situation spécifique, ou choisi par l'utilisateur.

Le principe de style se rencontre plus dans le domaine des interfaces homme-machine et du traitement du langage naturel [88, 99, 102]. Néanmoins, des travaux se sont intéressés à une

représentation plus concise des résultats de requêtes dans les cas de réponses importantes en volume [36, 50].

Le principe de relation évoque une adéquation maximale entre les enregistrements résultats et la requête. À moins 1) d'une différence de représentation entre requêtes et données, et 2) d'une transformation ambigüe vers l'espace où la comparaison s'effectue, aucun problème ne se pose quant à ce principe. Les critères d'une requête SQL sont en effet des valeurs possibles d'attributs.

Sadek [124] distingue six classes de réponses coopératives « qu'il serait appréciable d'obtenir d'un évaluateur de requêtes » [15, p. 6]. Les réponses fournies par les systèmes coopératifs s'inscrivent dans une ou plusieurs de ces classes :

1. les réponses complétives, qui fournissent plus d'information que la réponse valide minimale ;
2. les réponses conditionnelles, relevant du principe de qualité, qui précisent les conditions sous lesquelles la réponse est valide ;
3. les réponses incomplètes, dues à une contrainte de confidentialité sur les données ;
4. les réponses intensionnelles, qui précisent les caractéristiques partagées par les résultats au lieu de présenter une liste de résultats ;
5. les réponses correctives, qui justifient une absence de résultats par une explication ; ce type de réponses met en évidence une hypothèse de l'interrogateur responsable de l'échec de la requête. L'hypothèse, supposée vraie par l'interrogateur mais fausse en réalité, est souvent désignée par le terme de « présupposition » ;
6. les réponses suggestives, qui suggèrent une autre réponse que celle naturellement induite par la requête.

Par nature, l'interrogation des résumés est intensionnelle vis-à-vis des enregistrements de la base de données puisqu'un résumé offre une description d'enregistrements.

Le test de correspondance utilisé lors de l'évaluation d'une requête (voir section 2.2.2) permet, sans traitement supplémentaire, de déterminer les « raisons » de l'échec, c'est-à-dire les critères non satisfaits de la requête. Rappelons que le test de correspondance indique s'il est possible de trouver des résultats pour la requête en cours dans le sous-arbre dont la racine est soumise au test. Si cette possibilité est nulle, l'exploration du sous-arbre n'est pas entreprise. C'est à ce cas que nous nous intéressons ici. Les critères non satisfaits peuvent être mis en évidence, sous une forme dépendante de l'application, lors de la présentation des résultats.

L'interrogation des résumés s'inscrit également dans la classe des réponses suggestives grâce aux mécanismes de modification ci-après, déclenchés en l'absence de résultats.

4.3.2 Principes de la modification de requêtes

La modification intervient lorsque l'exploration ne peut plus progresser au-delà d'un nœud z qui ne décrit aucun résultat pour la requête en cours d'évaluation. Dans ce cas, la procédure de sélection (section 2.2) échoue pour le sous-arbre de z : la liste des résumés sélectionnés reste vide car certains descripteurs requis sont absents de z, ce résumé étant alors considéré comme un *point d'échec*. En l'absence d'information à propos de tels résultats, l'utilisateur ne peut se fier qu'à son intuition pour soupçonner leur existence.

La première stratégie que nous proposons, dite de **substitution de requêtes** (section 4.3.4), consiste à remplacer les descripteurs absents par d'autres descripteurs, dans une limite fixée par la distance en section 4.3.3. On obtient alors une nouvelle requête $\mathcal{Q}^*$, dite requête de substitution. Cependant, comme le montre l'exemple 14 ci-après, aucune garantie ne peut être donnée quant à l'existence de résultats pour $\mathcal{Q}^*$. Une possibilité de substitution est liée aux variables linguistiques : les descripteurs absents sont remplacés par les descripteurs les plus proches, ce qui nécessite la définition d'un ordre sur le domaine de la variable concernée.

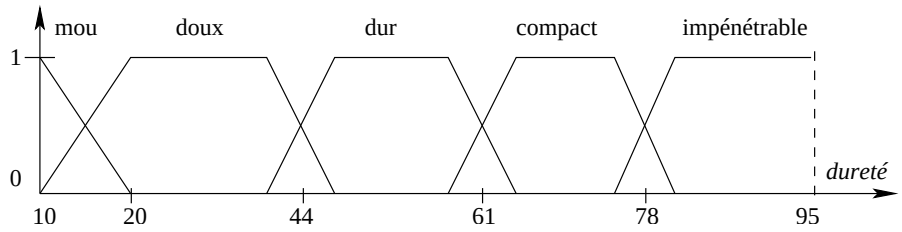


Figure 4.5 – Variable linguistique définie sur le domaine de l'attribut `dureté`

Exemple 14 :
Considérons une requête $\mathcal{Q}_1$ de caractérisation $\mathcal{C}_1 = \{\{dur\}\}$ sur l'attribut `dureté` *et un résumé z_0 tel que z_0.*`dureté` $= 1.0/$*mou. L'évaluation de $\mathcal{Q}_1$ sur z_0 ne permet pas de poursuivre la recherche dans le sous-arbre z_0. La substitution au niveau de z_0 remplace le critère « dur » par d'autres critères, par exemple « doux » et/ou « compact » (voir figure 4.5). Soit $\mathcal{Q}_1^*$, de caractérisation $\mathcal{C}_1^* = \{\{doux, compact\}\}$, la requête substituée à $\mathcal{Q}_1$. $\mathcal{Q}_1^*$ échouant également pour z_0, le sous-arbre z_0 ne sera pas visité.*

Dans l'éventualité où le domaine n'est pas ordonné, une autre possibilité, plus générale qu'un ordre naturel sur un domaine, consiste à définir explicitement une matrice de substitutions pour chaque variable. Les poids affectés aux permutations permettent alors un traitement plus fin des résultats en privilégiant certaines substitutions parmi celles possibles. La matrice de substitution définit implicitement une distance entre termes linguistiques. Elle est applicable aux domaines naturellement ordonnés et aux domaines non ordonnés. Le tableau 4.3 donne un exemple de matrice de substitution pour laquelle on suppose qu'une dureté supérieure est toujours préférable (le poids associé est alors supérieur). Ainsi, lorsque deux substitutions d_1 et d_2 existent pour un même descripteur d, la substitution la plus proche de la borne supérieure du domaine (d_2) est privilégiée. Ceci se traduit par le fait que les résultats des requêtes substituées « héritent » du poids de la substitution correspondante.

Dans tous les cas, un descripteur d remplacé par d^* au niveau d'un nœud z ne peut plus servir de substituant à un autre descripteur dans le sous-arbre de racine z. En effet, l'échec de d au résumé z garantit qu'il n'existe pas de résumé, dans le sous-arbre de racine z, où d serait positivement valué dans l'évaluation de la requête. Ceci est une conséquence de l'ordre partiel sur les résumés : d n'apparaît pas dans l'intension de z parce qu'aucun descendant de z ne présente cette étiquette (voir la section 1.5.2).

Exemple 15 :
Soit la requête $\mathcal{Q}_2$ de caractérisation $\mathcal{C}_2 = \{\{doux, dur\}\}$ sur l'attribut `dureté`*. L'évaluation de $\mathcal{Q}_2$ sur le résumé z_0 (de l'exemple 14) échoue car mou $\notin$ {doux, dur}. En appliquant la substitution de « doux », on obtient {mou, dur}. De même, « dur » conduit à {doux, compact}. La caractérisation à l'issue des substitutions est {mou, doux, dur, compact}. Or, on sait que $\mathcal{Q}_2$ a déjà échoué ; les descripteurs « doux » et « dur » ne fourniront donc pas de résultat. Au final, la requête $\mathcal{Q}_2^*$ substituée à $\mathcal{Q}_2$ est de caractérisation $\mathcal{C}_2^* = \{\{mou, compact\}\}$.*

Table 4.3 – Matrice de substitution pour l'attribut `dureté`

	mou	*doux*	*dur*	*compact*	*impénétrable*
mou	0.0	0.8	0.0	0.0	0.0
doux	0.5	0.0	0.8	0.0	0.0
dur	0.0	0.5	0.0	0.8	0.0
compact	0.0	0.0	0.5	0.0	0.8
impénétrable	0.0	0.0	0.0	0.8	0.0

La deuxième stratégie, dite de *modification guidée* (section 4.3.5), est plus flexible car elle est guidée par la hiérarchie de résumés. Les résultats sont cette fois de nouvelles requêtes plutôt que des résumés. Elle est adaptée à un mode interactif où l'absence de résultat conduirait à une ou plusieurs requêtes alternatives offrant chacune une garantie de résultat.

4.3.3 Distance entre requêtes

L'un des principes de la modification décrite en section 4.3.2 est qu'un test de correspondance négatif sur un résumé déclenche la substitution du critère non satisfait. Les exemples 14 et 15 montrent que cette substitution peut en entraîner une autre, elle-même susceptible d'échouer.

Exemple 16 :
Reprenons z_0 tel que $z_0.\texttt{dureté} = 1.0/\text{mou}$ avec $\mathcal{Q}_3$ de caractérisation $\mathcal{C}_3 = \{\{\text{impénétrable}\}\}$. $\mathcal{Q}_3$ subit une première modification ; on a $\mathcal{Q}_4$ avec $\mathcal{C}_4 = \{\{\text{compact}\}\}$, qui échoue également. On a successivement $\mathcal{Q}_5$ avec $\mathcal{C}_5 = \{\{\text{dur}\}\}$, puis $\mathcal{Q}_6$ avec $\mathcal{C}_6 = \{\{\text{doux}\}\}$ et $\mathcal{Q}_7$ avec $\mathcal{C}_7 = \{\{\text{mou}\}\}$ avant d'avoir un test de correspondance positif, mais un « mauvais » résultat.

Afin d'éviter une série de modifications conduisant à des résultats inappropriés, nous introduisons une mesure de distance entre requêtes. Cette distance représente la proximité des résultats fournis avec la requête initiale et correspond au nombre de substitutions effectuées sur le chemin d'un résumé feuille. Pour respecter le principe de pertinence (section 4.3.1), il est nécessaire d'imposer une limite à cette distance.

Admettons qu'une requête $\mathcal{Q}$ puisse être associée à une chaîne de bits S, de longueur fixe, dans laquelle un bit marque la présence (ou l'absence) du descripteur associé à cette position dans la requête (figure 4.6). On suppose que l'assignation des positions aux descripteurs est invariable une fois fixée.

Définition 6 (Distance entre requêtes) : *Soient $\mathcal{Q}$ et $\mathcal{Q}^*$ deux requêtes respectivement associées aux chaînes de bits S et S^*. La distance entre $\mathcal{Q}$ et $\mathcal{Q}^*$, notée $d(\mathcal{Q}, \mathcal{Q}^*)$, est le nombre de bits non nuls de $S \oplus S^*$ (S XOR S^*). Ce nombre rend compte des modifications (insertions ou suppressions de descripteur) nécessaires pour obtenir $\mathcal{Q}^*$ à partir de $\mathcal{Q}$. La distance ainsi définie, qui est une distance de Hamming [77], satisfait les propriétés suivantes :*

1. $d(\mathcal{Q}, \mathcal{Q}) = 0$
2. $\mathcal{Q} \neq \mathcal{Q}^* \Rightarrow d(\mathcal{Q}, \mathcal{Q}^*) > 0$
3. $d(\mathcal{Q}, \mathcal{Q}^*) = d(\mathcal{Q}^*, \mathcal{Q})$
4. $d(\mathcal{Q}, \mathcal{Q}^*) < \sum_{A_i \in \mathcal{C}} |D_{A_i}|$ *où D_{A_i} est l'ensemble des descripteurs de la variable linguistique sur l'attribut $\mathcal{A}_i$.*

Toutefois, cette distance n'est pas suffisamment fine pour permettre un processus automatique. Il peut s'avérer nécessaire que l'utilisateur guide la suite de la recherche : en effet, la distance ci-dessus place au même niveau des requêtes $\mathcal{Q}_1$ et $\mathcal{Q}_2$ par rapport à une requête $\mathcal{Q}$ sans distinguer la *proximité* des étiquettes ou une similarité sémantique. Par exemple, si une requête porte sur des matériaux à température décrite par « bas » (voir figure 1.2), des requêtes modifiées pour rechercher respectivement « froid » et « élevé » sont considérées équivalentes.

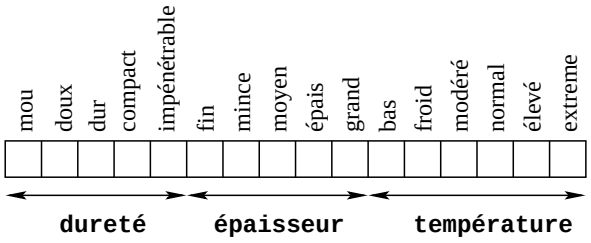


Figure 4.6 – Exemple d'assignation de positions sur une chaîne de bits

Exemple 17 :
Considérons les requêtes $\mathcal{Q}_1$ et $\mathcal{Q}_2$ avec une caractérisation impliquant les attributs `épaisseur` *et* `dureté` *: $\mathcal{C}_1$ ={{mince, moyen}, {dur, compact}} et $\mathcal{C}_2$ ={{mince, épais}, {dur, compact}}. En reprenant les assignations de la figure 4.6, on a :*

$$\begin{aligned} \mathcal{Q}_1 &\rightarrow \texttt{0011001}\mathbf{01}\texttt{0000000} \\ \mathcal{Q}_2 &\rightarrow \texttt{0011001}\mathbf{10}\texttt{0000000} \\ d(\mathcal{Q}_1, \mathcal{Q}_2) &= 2 \end{aligned}$$

Cette distance représente simplement le nombre de modifications de critère nécessaires pour passer d'une requête à une autre. Elle ne tient pas compte du sens des critères. Une distance plus adaptée doit donc être trouvée pour permettre un processus automatique basé sur des distances.

4.3.4 Algorithme de substitution de requêtes

Cette section présente la procédure de modification des requêtes dans l'algorithme 2. L'algorithme correspondant est une variante de l'algorithme 1 (section 2.2.3) à laquelle sont intégrés les principes de la modification évoqués plus haut.

Le parcours de la hiérarchie de résumés donne lieu à un test de correspondance à chaque nœud visité. Les résultats de ce test reprennent la notation déjà utilisée :

- *exacte* indique que le résumé examiné satisfait, pleinement et sans ambigüité, la requête ;
- *excès* indique plutôt qu'il est possible de trouver des résultats en explorant le sous-arbre, ce qui se traduit par un appel récursif de la fonction *Sel-Mod*.

Les informations disponibles (variables linguistiques ou matrice de substitution) sont exploitées à chaque fois que le test est négatif. Elles permettent à la fonction *Modifier* de constituer une requête substituée à $\mathcal{Q}_{ref}$, la requête originale exprimée par l'utilisateur. $\mathcal{Q}_{ref}$ est utilisée par la fonction *Modifiable* pour garantir que la propriété 4 (section 4.3.3) sera toujours vérifiée après la modification. La requête en cours d'évaluation, notée $\mathcal{Q}$ et initialement égale à $\mathcal{Q}_{ref}$, reste dans la limite d'une valeur de distance à fixer. Si la distance $d(\mathcal{Q}, \mathcal{Q}_{ref})$ est supérieure à ce seuil, la substitution conduirait à des résultats jugés trop éloignés de $\mathcal{Q}_{ref}$. Elle n'est donc pas effectuée.

Dans cet algorithme, la fonction Ajouter(z, L_{res}), qui enregistre le résumé z comme résultat, lui associe implicitement les informations de distance de la requête courante $\mathcal{Q}$ par rapport à la requête initiale $\mathcal{Q}_{ref}$. Notons que la liste de résultats (L_{res}) contient aussi bien des résultats de $\mathcal{Q}_{ref}$ que des résultats issus de requêtes de substitution. La liste est nécessairement triée afin de distinguer les meilleurs résultats, au sens de la proximité par rapport à la requête initiale.

Algorithme 2 Fonction Sel-Mod(z, $\mathcal{Q}$, $\mathcal{Q}_{ref}$)

```
L_res ← ⟨⟩
si Corr(z, Q) = excès alors
   pour chaque résumé z_fils fils de z faire
      L_res ← L_res+ Sel-Mod(z_fils, Q, Q_ref)
   fin pour
sinon
   si Corr(z, Q) = exacte alors
      Ajouter(z, L_res)
   sinon {pas de correspondance, mais le résumé peut être acceptable}
      si Modifiable(Q, Q_ref) = VRAI alors
         Q* = Modifier(Q, z)
         L_res ← L_res+ Sel-Mod(z, Q*, Q_ref)
      fin si
   fin si
fin si
retourner L_res
```

4.3.5 Modification guidée par les résumés

Lorsqu'une requête échoue, l'évaluation permet de détecter les *raisons* de l'échec, une *raison* désignant un attribut requis. Il suffit de mettre en évidence les attributs requis dont les descripteurs n'apparaissent pas dans le sous-espace exploré. Ces attributs ne sont pas nécessairement les mêmes pour les différents résumés où la requête échoue. Pour chacun de ces points d'échec, la substitution appropriée qui est effectuée peut échouer à son tour.

Pendant une recherche-sélection, si la même requête $\mathcal{Q}_0$ échoue pour tous les fils d'un nœud z_0, on peut considérer z_0 comme la meilleure approximation d'une réponse à $\mathcal{Q}_0$ dans la branche menant à z_0. L'exploration aboutissant à un résumé résultat se fait de proche en proche (figure 4.7-a), par tests successifs des parents. L'exploration apparaît comme un raffinement progressif en regard de chaque résumé sélectionné. Chaque parent d'une feuille résultat étant une généralisation du résultat, on considère le parent comme une approximation du résultat. Par exemple, si un résumé z_1 fils de z_0 est un résultat d'une requête $\mathcal{Q}_0$, une hiérarchie dans laquelle z_1 serait absent (figure 4.7-b) présentera z_0 comme résultat.

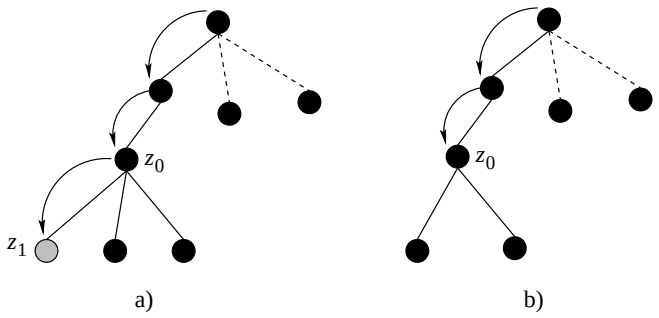


Figure 4.7 – Accès de proche en proche à un résultat

Du point de vue des requêtes, une modification implicite de la requête $\mathcal{Q}_0$ est effectuée sur la base de l'hypothèse que la proximité des résultats est équivalente à la proximité des requêtes. C'est ainsi que les caractéristiques du résumé approché (z_0 dans cet exemple) sur les attributs requis dans la requête initiale servent à construire une requête alternative.

Les requêtes semblables à $\mathcal{Q}_1$ (c'est-à-dire considérées comme proches de $\mathcal{Q}_0$ et issues de points d'échec), offrent une garantie de résultat, ce qui n'était pas le cas pour $\mathcal{Q}_0$.

Exemple 18 :

Soit la requête $\mathcal{Q}_0$ avec une caractérisation $\mathcal{C}_0$ = {{mince, moyen}, {dur, compact}} sur les attributs `épaisseur` *et* `dureté`. *Soit un résumé z_0 tel que z_0.`épaisseur` = ⟨1.0/mince + 1.0/épais⟩ et z_0.`dureté` = ⟨1.0/doux + 0.8/dur⟩ avec deux fils z_1 et z_2 :*

- z_1.épaisseur = ⟨*1.0/mince*⟩ *et* z_1.dureté = ⟨*1.0/doux*⟩ *;*
- z_2.épaisseur = ⟨*1.0/épais*⟩ *et* z_2.dureté = ⟨*0.8/dur*⟩.

Aucun des résumés z_1 et z_2 ne correspondant à la requête dans son sens strict, z_0 constitue le résultat approché fourni comme réponse pour ce qui est du sous-arbre de racine z_0.

La requête $\mathcal{Q}_0$ est remplacée par $\mathcal{Q}_1$, dont la caractérisation est donnée par z_0, toujours sur les attributs épaisseur *et* dureté*. On a $\mathcal{C}_1$ = {{mince, épais}, {doux, dur}}.*

Toutefois, ce fonctionnement requiert de déterminer, par l'algorithme 1, que la requête n'a pas de résultat et, par la suite, de rechercher les points d'échec afin de déduire les requêtes modifiées. L'algorithme 3 supprime la nécessité de deux explorations en constituant deux listes résultats : celle des résumés sélectionnés (L_{res}) et celle des résumés à la base de requêtes modifiées (L_Q), la dernière n'étant utile que si la première est vide.

L'algorithme effectue le même parcours que l'algorithme 1. Il utilise la correspondance, notée *myCorr*, du résumé z à la requête. Si le résumé répond à la requête, il est inséré dans la liste des résultats (L_{res}). Si une exploration du sous-arbre est nécessaire pour trouver d'éventuels résultats (myCorr = *excès*), elle est effectuée par appel récursif de la procédure *Sel-Guidée*. À l'issue de cette exploration, z est inséré dans la liste L_Q si aucun résultat n'est trouvé. Au final, *Sel-Guidée* fournit les listes L_{res} et L_Q.

Algorithme 3 Procédure Sel-Guidée(z, myCorr, $\mathcal{Q}$)

```
L_res ← ⟨⟩
L_Q ← ⟨⟩
si myCorr = excacte alors
   Ajouter(z, L_res)
sinon
   si myCorr = excès alors
      pour chaque résumé z_fils fils de z faire
         corr_f = Corr(z_fils, Q)
         L_res ← L_res+ Sel-Guidée(z_fils, corr_f, Q)
      fin pour
      si L_res = ⟨⟩ alors
         Ajouter(z_fils, L_Q)
      fin si
   fin si
fin si
retourner L_res, L_Q
```

4.3.6 Expression des résultats

La modification de requêtes, telle qu'elle est décrite ci-dessus, rend possible l'obtention de résultats issus de plusieurs requêtes modifiées différentes. Ces requêtes sont plus ou moins proches de la requête initiale. Il devient possible d'effectuer un classement des résultats et donc d'exprimer une préférence de certains résultats par rapport à d'autres. Il semble naturel que le critère de classement des résultats soit la distance des requêtes modifiées relativement à la requête utilisateur. Ainsi, les résultats se voient associer la distance de la requête dont ils sont un résultat.

On notera que la modification apparaît comme un relâchement des contraintes de la recherche car la nouvelle requête obtenue est plus générale. Cependant, chaque substitution d'une requête $\mathcal{Q}$ par une requête $\mathcal{Q}^*$ est locale comme l'explicite l'exemple 19 ci-après. Le caractère local de la modification se justifie par deux raisons :

- le fait que l'échec survient à un point précis du parcours, en l'occurrence un résumé z. La modification a pour but de *réparer* la requête au point d'échec afin de permettre la poursuite de l'exploration de la hiérarchie. Il n'est pas utile d'étendre la portée de la modification au-delà de l'espace de données décrit par z car l'objectif poursuivi est atteint ;
- le nombre des échecs qui, sans ce caractère local, ferait perdre en efficacité. L'algorithme de recherche tire en effet parti des coupures de branches : plus le nombre de coupures est élevé, meilleure est l'efficacité de la recherche. Une coupure correspond, dans l'optique de la modification, à un point d'échec, et donc, à une requête de substitution supplémentaire. Si la contrainte de localité était supprimée, toutes les requêtes de substitution seraient exécutées sur l'ensemble de la hiérarchie, ce qui dégraderait les performances du traitement.

Exemple 19 :

Considérons une requête initiale $\mathcal{Q}_0$ adressée à une hiérarchie de racine z_0 et deux points d'échec arbitraires, z_1 et z_2. Les requêtes de substitution $\mathcal{Q}_1$ et $\mathcal{Q}_2$ sont alors composées pour chacun de ces points.

On distingue, suivant la requête dont ils sont un résultat, trois types de résumés sélectionnés : ceux de l'arbre z_0, ceux du sous-arbre z_1 et ceux du sous-arbre z_2.

Les résumés sélectionnés par $\mathcal{Q}_1$ ne se retrouveront que *dans le sous-arbre de sommet z_1 même s'il existe ailleurs dans l'arbre, des résumés qui auraient été sélectionnés si $\mathcal{Q}_1$ avait été adressée à toute la hiérarchie (à partir de z_0). Il en est de même pour $\mathcal{Q}_2$.*

En résumé, la modification s'articule comme suit avec l'algorithme 2.

1. évaluation de la requête $\mathcal{Q}$ (par l'algorithme 1, section 2.2.3) ;
2. si absence de résultat, exécution de la modification ;
   (a) exécution de la modification ;
   (b) choix, par la mesure de distance ou par interaction, d'une requête de substitution $\mathcal{Q}^*$;
   (c) évaluation de $\mathcal{Q}^*$ (retour au point 1) ;
3. sinon, expression des résultats.

Avec l'algorithme 3, la modification est implicite et systématique. On a :

1. évaluation de la requête initiale $\mathcal{Q}$ (par l'algorithme 3) ;
2. si L_{res} est vide, construire les requêtes de substitution à partir des éléments de L_Q ;
   (a) choix par interaction, d'une requête de substitution $\mathcal{Q}^*$;
   (b) évaluation de $\mathcal{Q}^*$ (l'algorithme 1, section 2.2.3) ;
   (c) expression des résultats ;
3. sinon, expression des résultats.

4.4 Utilisation d'un vocabulaire personnalisé

L'un des points forts des résumés SAINTETIQ mis en avant jusqu'ici a été l'utilisation d'un unique vocabulaire partagé par une communauté d'utilisateurs. Ce vocabulaire, constitué de variables linguistiques, définit pour chaque attribut l'ensemble des termes linguistiques utilisés pour décrire des valeurs du domaine d'attribut de manière plus générale, plus grossière. Dans le cas où les résumés sont construits pour une communauté d'utilisateurs, il est souhaitable que le sens affecté aux termes soit consensuel. Cependant, on considère ici l'éventualité où le vocabulaire utilisé pour spécifier les caractéristiques des objets relationnels n'est pas celui utilisé pour la construction des résumés. Par exemple, ce pourrait être le cas si les résumés construits à l'intention d'une communauté sont utilisés dans une autre communauté.

On admet que ce vocabulaire personnalisé $\mathcal{V}^*$ adopte la même forme que le vocabulaire initial $\mathcal{V}$: il est constitué de variables linguistiques sur les domaines d'attributs. On fait également l'hypothèse que $\mathcal{V}^* \neq \mathcal{V}$. Les variables linguistiques qui composent respectivement $\mathcal{V}^*$ et $\mathcal{V}$ sont donc nécessairement différentes, sur au moins un domaine d'attribut.

L'utilisation d'un vocabulaire différent implique une mise en correspondance des termes issus des vocabulaires $\mathcal{V}^*$ et $\mathcal{V}$. Cette section traite de cette correspondance et de ses implications en fonction de la nature de la requête et de celle des résultats attendus.

Motivations Les raisons qui nous poussent à envisager l'usage d'un vocabulaire personnalisé lors de l'interrogation sont doubles. D'une part, le chapitre précédent a montré que les langages d'interrogation flexible basés sur les sous-ensembles flous permettent la définition « au vol » de termes linguistiques, de variables linguistiques, de qualificateurs, etc. Ces propositions, qui introduisent de la flexibilité dans les critères de sélection, les langages d'interrogation et, par prolongation, les bases de données relationnelles, sont intrinsèquement dynamiques sous l'aspect linguistique. Or, les hiérarchies de résumés, bien que dynamiques (voir [125]) vis-à-vis de la maintenance de la structure, restent intrinsèquement statiques vis-à-vis du vocabulaire de construction. Les variables linguistiques incluses dans les connaissances de domaine doivent effectivement être fixées avant la construction des résumés (voir section 1.2). Le travail de cette section est donc un moyen de rapprocher ces deux natures opposées.

D'autre part, les systèmes de personnalisation partagent avec l'interrogation flexible le même intérêt pour les préférences. Dans ces systèmes, des données propres à un utilisateur sont conservées ; on parle alors du « profil » de l'utilisateur. L'usage du profil permet d'adapter les réactions du système à l'utilisateur, offrant ainsi un comportement « personnalisé ». Le profil spécifie bien sûr les préférences de l'utilisateur, mais aussi toute autre information pertinente pour le contexte d'application. En particulier, un profil peut contenir le vocabulaire de l'utilisateur, exprimé sous la forme de variables linguistiques. Dès lors, l'hypothèse que ce vocabulaire correspond à celui de la hiérarchie de résumés ne peut être supposée vraie dans le cas général.

4.4.1 Principes

Soit un attribut A, d'une relation R, dont le domaine D_A est continu. Supposons que la variable linguistique sur A utilisée pour la construction des résumés est définie par la partition P_1 en figure 4.8. En l'état actuel, le processus d'interrogation n'exploite pas le caractère graduel des sous-ensembles flous définis par les termes des variables linguistiques. Les termes linguistiques sont donc assimilés à leur support sur le domaine D_A.

En première approche, utiliser une partition P_2 de D_A (voir figure 4.9) différente de P_1 pour *retrouver* ou *sélectionner* des objets sur la base de leur description par un terme de P_1 équivaut à une mise en correspondance des deux partitions (voir fig. 4.10). Considérons une requête $\mathcal{Q}^*$ sur un seul attribut A dont le critère de sélection, monovalué, spécifie un descripteur l de P_2 comme clé de recherche. $\mathcal{Q}^*$ s'exprime, si R est la relation résumée, par :

```
SELECT * FROM R WHERE A = l;
```

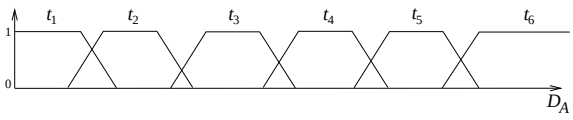


Figure 4.8 – Partition P_1 du domaine D_A

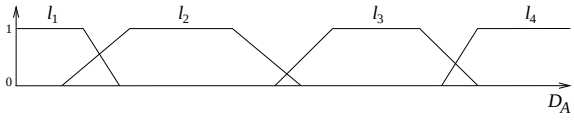


Figure 4.9 – Partition P_2 du domaine D_A

Considérons également que l'expression des résumés est faite grâce à la partition P_1. Sur le domaine D_A, le terme l (de P_2) dénote une valeur dans l'intervalle $]a, b[$ tandis que le terme t (de P_1) décrit l'intervalle $]c, d[$ (voir fig. 4.10).

Trois situations distinctes où les intervalles se recouvrent peuvent survenir :

1. l est entièrement couvert par t ;
2. t est couvert, partiellement ou totalement, par l ;
3. les intervalles du domaine définis par t et l sont identiques (leurs supports sont égaux).

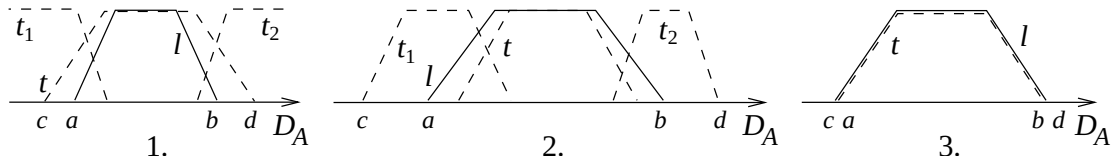


Figure 4.10 – Situations relatives des descripteurs t et l sur un même domaine

Vu que nous traitons spécifiquement du cas où le vocabulaire est différent, la situation 3 est moins plausible que les autres. Elle est également plus simple : si les supports sont égaux, les termes t et l sont équivalents et tous les principes de l'interrogation approchée, exposés dans le chapitre 2 restent inchangés. Dans les autres situations, une réécriture de la requête s'impose pour tenir compte du vocabulaire initial des résumés. Nous proposons une substitution dont le but est capturer tous les tuples décrits par le terme l du vocabulaire personnalisé. Ainsi, on substitue à l la plus petite union de termes de $\mathcal{V}$ dont le support contient celui de l. La requête $\mathcal{Q}^*$ initiale devient :

dans la situation 1, $\mathcal{Q}_1$: `SELECT * FROM R WHERE` A `=` t

dans la situation 2, $\mathcal{Q}_2$: `SELECT * FROM R WHERE` A `IN (`t`,` t_1`,` t_2`)`

La réécriture de la requête ne pose en elle-même pas de problème au niveau de la sémantique puisque les requêtes réécrites sont des requêtes valides. Cependant, elle soulève une question importante liée aux intervalles induits sur le domaine D_A :

« comment distinguer dans $]c, d[$ les valeurs qui ne correspondent qu'à $]a, b[$? »

Autrement dit, lorsqu'une requête précise l comme critère, l'on sait que ce critère se réécrit par t, t_1 et t_2, en accord avec notre politique de réécriture. Mais les objets décrits par t, t_1 et t_2 ne sont pas tous nécessairement décrits par l. Ce problème, traité en section 4.4.2, se pose également lorsque l'interrogation est intégrée à un SGBD (section 6.1.1).

La difficulté de la mise en correspondance des deux partitions réside dans le cas 2. D'une manière générale, on note qu'une requête réécrite contient un nombre plus grand de termes. Dans le cas où la requête fait apparaître un seul attribut, le critère de sélection est disjonctif et s'exprime sous la forme $A = l_1 \vee l_2 \vee \ldots \vee l_n$. La réécriture donne lieu à une substitution identique à celle évoquée plus haut. La nouvelle requête reste disjonctive, avec un ensemble de termes a priori plus grand puisque la situation 2 prévaut. Dans l'éventualité d'une sélection multi-attributs (par exemple, $A_1 = l_1 \wedge A_2 = l_2$), la réécriture reste valable puisque la substitution est plus générale que la requête initiale. Tous les résultats sont donc sélectionnés.

Cependant, dès qu'une substitution intervient, des éléments étrangers à l'intervalle $]a, b[$ qu'implique un terme l (voir figure 4.10) peuvent être présents parmi les résultats. Il suffit pour cela que ces éléments fassent également partie d'un autre sous-ensemble flou de la réécriture, t_1 par exemple dans le cas 2 de la figure 4.10. La présence de résultats non sollicités est acceptable 1) s'il est possible de distinguer ces « faux positifs » ou 2) si l'on admet que le résultat de cette interrogation (avec un autre vocabulaire que celui de la construction des résumés) peut être approximatif. La propriété de complétude de la sélection est alors maintenue (tous les résultats sont sélectionnés) mais pas la propriété d'adéquation qui spécifie que la réponse ne contient aucun résultat non valide.

4.4.2 De la possibilité de distinguer les faux positifs

Déterminer si un élément X de l'ensemble des résultats d'une interrogation est un résultat non sollicité revient à déterminer si la valeur $X.A$ appartient à l'intervalle $[a, b]$ sur le domaine D_A induit par le critère de recherche l sur l'attribut A. Sachant que ces valeurs d'attribut ne sont pas disponibles au sein de la hiérarchie de résumés, l'accès à la base de données devient nécessaire. En effet, un résumé ne conserve, pour une étiquette linguistique donnée, qu'une valeur (le degré de satisfaction maximale) associée au domaine de l'attribut concerné. Cette valeur réelle qualifie la description de tous les objets du résumé par l'étiquette linguistique. C'est une valeur d'ensemble qui n'est pas spécifique à l'élément X.

Si l'accès à la base n'est pas possible, on admettra que les faux positifs sont inhérents à cette approche de l'utilisation d'un vocabulaire différent. Dans le cas contraire, ils sont écartés lors de l'étape de filtrage, post-traitement de la recherche (voir section 6.1.1, chapitre 6). Une réponse

précise à une requête présente des enregistrements comme résultats. L'accès à la base de données implique donc le filtrage, qui résout le problème des faux positifs. Nous nous intéressons ici aux requêtes dont les réponses sont imprécises, c'est-à-dire exprimées par des résumés.

4.4.3 Réponses approchées par imprécision

Considérons une requête dont le critère de sélection ne comporte qu'un terme, par exemple, $\mathcal{Q}^*$: `SELECT * FROM R WHERE` $A = l$. l est un terme du vocabulaire d'interrogation $\mathcal{V}^*$, auquel sont substitués les termes t_1 à t_n du vocabulaire $\mathcal{V}$ de construction des résumés. On obtient $\mathcal{Q}$: `SELECT * FROM R WHERE` A `IN` $(t_1, \ldots, t_n)$.

Les termes t_1 à t_n apparaissent dans la substitution car les intervalles qu'ils définissent sur le domaine D_A sont intersectés par celui du terme l (cas 2 de la figure 4.10). Puisque les variables linguistiques sont définies a priori et non en fonction des données résumées, il n'existe, à l'échelle d'un terme t_i, aucune garantie qu'un résultat à la requête corresponde effectivement au terme l. En effet, une valeur x du domaine D_A appartenant au noyau de l n'est pas distinguée d'une valeur y, couverte non par l mais par l'un des termes de substitution (le terme t_3 sur la figure 4.11). Par contre, l'existence même de résumés résultats à la requête réécrite $\mathcal{Q}$ indique la *possibilité* de trouver des enregistrements correspondant exactement à l.

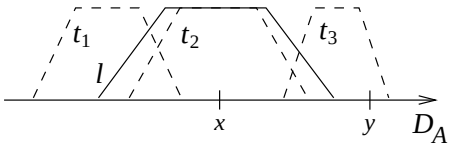


Figure 4.11 – Situations relatives d'un terme et de ses substitutions

Nous pouvons ici distinguer les cas de figure où une interrogation consécutive à une substitution de termes offre une garantie de correction du résultat vis-à-vis des enregistrements. Soit $\mathcal{R}_z$ l'ensemble des résumés formant une réponse approchée à la requête $\mathcal{Q}$. $\mathcal{R}_z$ est exprimé suivant le vocabulaire $\mathcal{V}$ de construction des résumés. Soit $\mathcal{R}_z^*$ l'expression de la réponse $\mathcal{R}_z$ suivant le vocabulaire $\mathcal{V}^*$.

4.4.3.1 Garantie sur les réponses vides

Dans les substitutions décrites précédemment, les propriétés de l'algorithme d'interrogation ne sont pas modifiées. En particulier, une réponse approchée non-vide implique nécessairement que des enregistrements correspondants existent car tout résumé dans une hiérarchie décrit des

données. Il n'est donc pas possible d'obtenir une réponse précise vide à partir d'une réponse approchée non-vide.

En conséquence, une réponse approchée vide garantit une réponse précise vide. Par contre, il n'est pas possible de garantir qu'une réponse approchée non-vide fournira une réponse précise non-vide. Ceci est dû à la réécriture des critères en critères plus généraux : $\mathcal{R}_z$ peut ne contenir que des faux positifs. La garantie sur les réponses vides est associée à la propriété de complétude.

4.4.3.2 Garantie sur les réponses non-vides

Pour garantir des réponses non-vides, il est nécessaire d'utiliser une réécriture « contrainte » : les termes de substitution doivent être plus sélectifs que les termes initiaux. Pour ce faire, le terme l (de $\mathcal{V}^*$) est remplacé par les termes t_i (de $\mathcal{V}$) tels que $t_i \subseteq l$ (figure 4.12-a). Le sens de l'inclusion des supports est inversé : c'est maintenant le terme l qui doit être plus général que le terme de substitution.

L'inconvénient ici est que les réponses ne sont pas toutes capturées. Il peut y avoir des réponses vides alors même que des enregistrements sont décrits par le terme l. Il suffit qu'aucun terme t_i du vocabulaire de réécriture $\mathcal{V}^*$ ne soit entièrement inclus dans l (figure 4.12-b). Contrairement à la précédente, cette approche garantit l'adéquation mais au détriment de la complétude. L'adéquation traduit le fait que dans le cas d'une réponse approchée vide, la réponse précise est aussi non-vide.

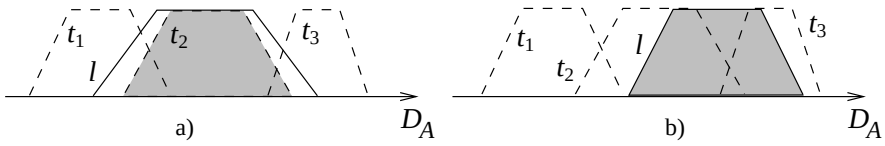


Figure 4.12 – Réécriture contrainte

4.4.3.3 Réduire l'incertitude

En résumé, on note que dans le cas général, le résultat de l'interrogation de résumés à l'aide d'un vocabulaire $\mathcal{V}^* \neq \mathcal{V}$ est caractérisé par une incertitude. Celle-ci porte, suivant l'option de réécriture adoptée, soit sur la complétude de l'interrogation (existence possible de faux négatifs), soit sur son adéquation.

Une solution pourrait être de conserver, pour chaque sous-ensemble flou de la variable linguistique sur l'attribut A, les valeurs minimales et maximales sur le domaine D_A atteintes par les données décrites par le terme. Ainsi, un sous-ensemble flou trapézoïdal est décrit par le quadruplet usuel (a, b, c, d) augmenté par le couple (v_{min}, v_{max}) (figure 4.13). Cette solution est partielle puisqu'elle ne permet toujours pas d'éviter la situation décrite par la figure 4.12-b. Néanmoins, elle permet de réduire les cas où les résultats sont teintés d'incertitude puisque l'intervalle $[v_{min}, v_{max}]$ est nécessairement contenu dans l'intervalle $]a, d[$. On tirerait ainsi parti des situations où $[v_{min}, v_{max}]$ est couvert par l. La figure 4.14 montre un exemple où la réécriture contrainte peut s'effectuer malgré l'absence d'inclusion du support de t_2 dans celui de l. D'autre part, sur le plan pratique, cette solution (du sous-ensemble flou « augmenté ») présente l'avantage d'être indépendante du volume des données résumées et de ne représenter qu'un volume supplémentaire de données relativement faible dans les connaissances de domaine.

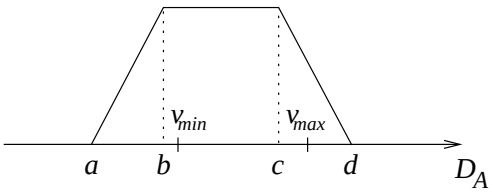


Figure 4.13 – Sous-ensemble flou trapézoïdal augmenté

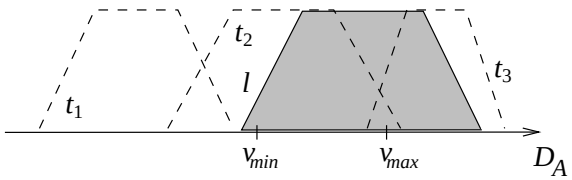


Figure 4.14 – Réécriture contrainte avec un sous-ensemble flou augmenté

Une autre façon de réduire l'incertitude consiste à conserver les degrés minimum et maximum d'appartenance aux résumés, déterminés descripteur par descripteur. Ces degrés permettent de circonscrire le support du sous-ensemble flou, de la même manière que la solution du sous-ensemble flou « augmenté » réduit l'intervalle $]a, d[$ à $[v_{min}, v_{max}]$.

4.5 Conclusion

L'interrogation des résumés est utilisée pour fournir une réponse approchée, exprimée par des résumés, à une requête. Le prototype implémentant cette interrogation a été présenté, de même que les expériences menées. Ces expériences ont permis d'évaluer les temps d'exécution de l'algorithme sur deux jeux de données. Il en ressort que l'existence et la caractérisation de données satisfaisant des critères est déterminée en des temps courts (41 ms en moyenne pour le plus volumineux des jeux de données). Notons que des optimisations de l'implémentation permettront d'améliorer les temps obtenus.

Le cas des requêtes sans réponse est discuté. L'absence de résultat est traitée par relaxation des critères de recherche, et donc par modification de la requête initiale. Partant du test d'appariement résumé-requête de l'algorithme d'interrogation, les attributs non satisfaits par le résumé sont identifiés sans traitement additionnel. Un nouvel algorithme, basé sur l'algorithme précédent, est proposé pour trouver des résumés proches de ceux désignés par la requête mais absents de la hiérarchie de résumés interrogée. Pour ce faire, l'algorithme exploite des informations supplémentaires fournies explicitement sous la forme de mesures de distances ou de matrices de substitution. En outre, les principes d'une modification implicite, sans information supplémentaire, sont discutés. L'algorithme correspondant est ensuite proposé. Il utilise les résumés de la hiérarchie pour garantir que des réponses existent pour les requêtes modifiées qu'il propose.

L'utilisation d'un vocabulaire différent de celui utilisé lors de la construction des résumés est également discutée. Ne sont traités ici que les interrogations ayant une réponse approchée. Les cas où la réponse est détaillée, exigeant un accès aux enregistrements, sont différés dans les chapitres suivants. On montre que le vocabulaire personnalisé introduit une incertitude sur les résultats en ce sens qu'il n'est pas possible d'assurer, pour toute réponse approchée, que tous les enregistrements résultats et seulement ceux-là, sont décrits par la réponse fournie.

CHAPITRE 5

Indexation de données

Introduction

Les modèles de bases de données actuels visent, abstraction faite des fonctionnalités dites « avancées » (réplication, reprise après pannes, concurrence, ...), le même objectif que les modèles précédents : stocker et gérer des données d'une manière sûre, cohérente et efficace. L'impératif d'efficacité a par le passé été motivé par les faibles quantités de mémoire disponibles, et les capacités de traitement limitées. Cet impératif revêt aujourd'hui une dimension supplémentaire avec le volume des bases de données. Les outils utilisés pour respecter la contrainte d'efficacité sont, d'une manière ou d'une autre, des index.

Un index est une structure de données construite à partir d'enregistrements d'une base de données et organisée de façon à permettre de localiser rapidement les enregistrements recherchés. L'index contient dans tous les cas des informations permettant de guider les recherches vers les données d'intérêt. Les bases de données étant rarement statiques, c'est-à-dire constituées une fois pour toutes, sans possibilité ou intention de modifications, la structure de données qu'est l'index est amenée à évoluer.

Tout l'enjeu des techniques d'indexation réside dans leur capacité à créer une structure de données dotée de propriétés les plus proches possibles de celles de la structure idéale dans le rôle d'index. Il s'agit notamment de sa taille, faible par rapport aux données indexées, de sa facilité d'implémentation et de maintenance, et de sa capacité à discriminer les bonnes réponses. Une autre propriété importante mais souvent négligée est la capacité à effectuer tout autre traitement dans des temps courts. Notons qu'un index offre l'avantage d'accélérer les recherches, mais il a aussi un surcoût lié à sa gestion, en particulier lors des mises-à-jour. Typiquement, une technique d'indexation ne met pas seulement en jeu une structure de données, mais aussi des algorithmes d'insertion, de modification, et de suppression au sein de l'index.

Dans ce chapitre, nous présentons des généralités sur les index existants afin de rapporter les résumés SAINTETIQ à ces techniques. La section 5.1 aborde les index monodimensionnels, présents dans la plupart des systèmes de gestion de bases de données actuels. La suite du chapitre traite des index multidimensionnels. Quelques techniques sont présentées, de même que

les propriétés permettant d'évaluer les index. Cette présentation permet de tirer des conclusions sur les caractéristiques partagées par les index et de montrer que les résumés SAINTETIQ en font partie.

5.1 Index monodimensionnels

Un index est dit monodimensionnel lorsque les données utilisées pour sa construction sont issues d'un unique domaine d'attribut D. La littérature sur ces index les classe dans différentes catégories suivant leurs propriétés. Les distinctions à la base de ces catégories sont abordées ci-dessous.

5.1.1 Index primaire et index secondaire

Lorsque la table de données est triée, l'attribut qui fixe l'ordre des enregistrements (sur le support de stockage) sert souvent de clé de recherche. Si tel est le cas, l'index est dit *primaire*, sinon il est qualifié d'*index secondaire*. Par définition, il ne peut exister qu'un index primaire par fichier séquentiel d'une relation. Il est à noter que le qualificatif « primaire » est parfois utilisé pour désigner un index construit sur une clé primaire de la relation. Cet usage semble abusif ; la terminologie pour un tel index basé sur un identifiant le qualifie d'index *unique*.

5.1.2 Index groupé et index non groupé

Un index de fichiers séquentiels peut maintenir les données triées de manière à regrouper physiquement les enregistrements correspondant à une même valeur. Il s'agit alors d'un index groupé (*clustered index*, figure 5.1) : l'ordre des enregistrements sur le support physique est identique à (ou très proche de) l'ordre des valeurs de la clé de recherche (également désignée par le terme « attribut d'indexation »). De même qu'il n'y a qu'un index primaire par relation, il ne peut y avoir qu'un index groupé par relation. Le coût d'utilisation de l'index varie de manière importante suivant cette caractéristique : la conservation de l'ordre séquentiel détermine la possibilité d'un parcours séquentiel lors des recherches par intervalle. Dans un tel parcours, plusieurs enregistrements sont disponibles en un accès disque au lieu d'un accès par enregistrement si le parcours devait être initié pour un index non groupé.

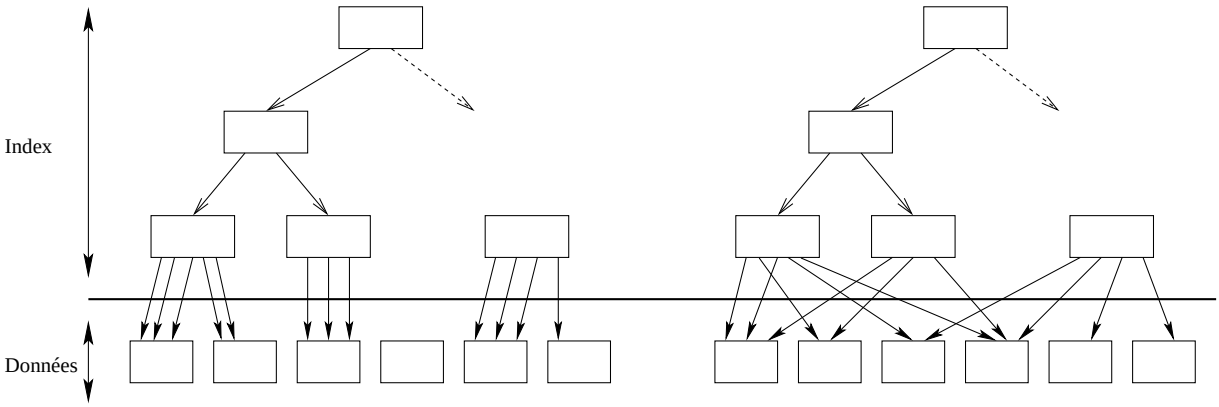


Figure 5.1 – Index groupé et index non groupé

5.1.3 Index dense et index non dense

Un index contient des informations sur les données qui guident les recherches. L'ensemble d'informations liées à une valeur de la clé de recherche constitue une entrée d'index. Typiquement, une entrée consiste en une valeur de l'attribut d'indexation et un pointeur vers un ou plusieurs enregistrements correspondants (voir la figure 5.2). Deux options de représentativité des entrées d'index par rapport aux données existent : il peut y avoir une entrée par valeur existante de l'attribut d'indexation. L'index, alors dit *dense*, peut être relativement important

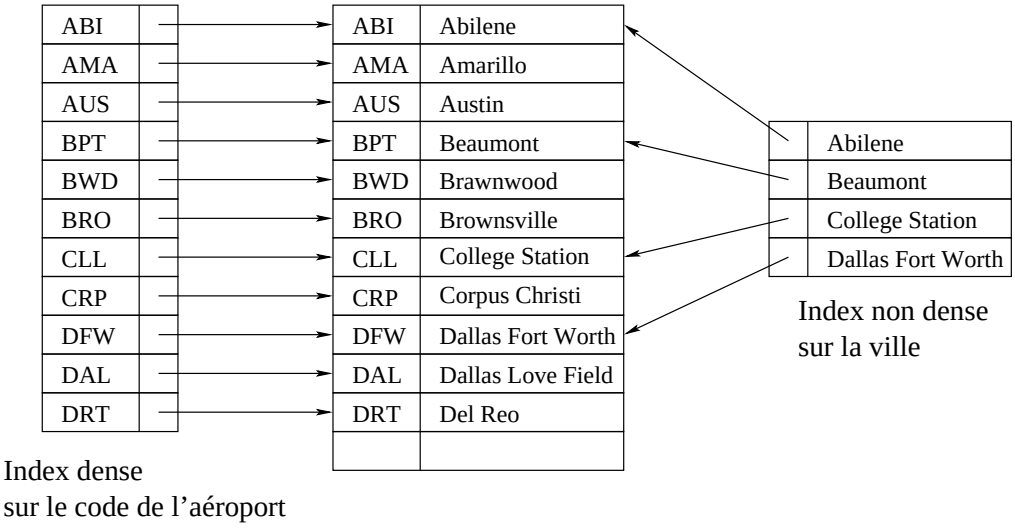


Figure 5.2 – Index dense et index non dense

en taille. L'autre option consiste à utiliser un index *non-dense* : une entrée d'index correspond implicitement à plusieurs valeurs de la clé de recherche, présentes ou non parmi les données. La contrepartie au volume plus faible est un traitement ultérieur en mémoire plus long. La figure 5.2 montre un exemple d'index dense et d'index non dense sur un fichier contenant les codes d'aéroports du Texas. Le traitement ultérieur en mémoire peut être la recherche séquentielle au sein d'un bloc de données à partir d'une position. Par exemple, la recherche de « Austin » s'effectue en deux étapes : la détermination d'une position du fichier où commencer la recherche, donnée par « Abilene », et la recherche séquentielle à partir de cette position.

Notons que tout index non-dense est nécessairement groupé (autrement la recherche deviendrait purement séquentielle) et que tout index secondaire est nécessairement dense.

Dans la suite de cette section sont présentées les méthodes d'indexation portant sur un attribut : les tables de hachage, l'index ISAM et les index de la famille de l'arbre B.

5.1.4 Tables de hachage

Les tables de hachage fournissent un accès direct à un enregistrement spécifique t grâce à une fonction f dite fonction de hachage. La fonction, qui a comme argument la valeur d'un attribut A (dit « champ de hachage ») de l'enregistrement, retourne une valeur Ax :

$$Ax = f(t.A) \ .$$

L'attribut A peut être la clé primaire ou n'importe quel autre attribut. À l'insertion de l'enregistrement, la valeur Ax est utilisée pour déterminer l'emplacement de l'enregistrement t. De même, Ax sert pour les recherches ; le même calcul est effectué et l'emplacement désigné reste le même.

La valeur Ax peut fournir directement l'adresse physique de t. Mais cette solution, dite de *hachage direct*, se révèle peu pratique et n'entre pas dans la catégorie des index [48] : le hachage direct n'autorise en effet qu'un champ de hachage par fichier de données. Dans une autre option, le *hachage indirect*, la valeur Ax désigne une entrée dans une table associée à une section physique (une ou plusieurs pages) du fichier indexé. On parle de *collision* lorsque les valeurs $Ax_1 = f(t_1.A)$ et $Ax_2 = f(t_2.A)$ d'emplacement calculées pour deux enregistrements t_1 et t_2 sont égales. Ce phénomène est intrinsèque au principe même du hachage. Il a fait l'objet de plusieurs travaux, notamment en cryptographie (voir [7, 123, 131]). L'ensemble de départ (ici, l'ensemble des valeurs possibles de l'attribut de hachage) a un cardinal plus grand que l'ensemble d'arrivée (l'intervalle des indices de la table) ; il est donc inévitable que des valeurs différentes aient des images identiques.

Figure 5.3 – Table de hachage

Dans le cas général, il n'est pas possible de garantir que le nombre d'enregistrements placés dans toute section n'excédera pas la capacité de l'emplacement. C'est pourquoi les index par hachage prévoient des zones de débordement où placer les enregistrements affectés à une section pleine. Cependant, ces zones de débordement réduisent l'efficacité de la méthode d'indexation : l'accès à une section est direct mais la recherche au sein de la section reste séquentielle.

Le hachage a comme avantage d'être très rapide. Mais la réorganisation physique nécessaire pour réduire les zones de débordement réserve les méthodes de hachage aux tables de données plutôt statiques et de faible taille. De plus, la séquence des valeurs de l'attribut de hachage n'est pas conservée sur le support de stockage dans le cas général, ce qui limite les critères de recherche aux critères d'égalité. Par exemple, la séquence de données Abilene - Amarillo - Austin - Beaumont - Brawnwood - Brownsville détermine la séquence de pages 1 - 2 - 3 - 4 - 1 - 2. Il est néanmoins possible de construire des fonctions de hachage qui garantissent que la séquence physique des enregistrements correspond à la séquence logique des valeurs d'attribut (voir [72] par exemple). Les recherches par intervalles de valeurs ne peuvent être supportées qu'à cette condition.

Les améliorations des tables de hachage (par exemple, [56]) visent spécifiquement le problème des collisions. Ces méthodes permettent de faire varier le nombre des emplacements et d'assigner dynamiquement des enregistrements aux emplacements qui optimisent la partie séquentielle des recherches.

On notera que toute méthode d'indexation monodimensionnelle par table de hachage garantit l'extraction de l'enregistrement recherché en un maximum de 2 accès disque (s'il n'y a pas de débordement) : un accès pour lire l'emplacement adéquat dans la table de hachage et un accès pour lire l'enregistrement.

5.1.5 ISAM

Un index ISAM (Indexed Sequential Access Method) [86] est un arbre de recherche général non-dense, donc groupé, conçu par IBM avant l'apparition des bases de données relationnelles. Il est adapté aux recherches par plages de valeurs (ou recherches par intervalles) dans un fichier séquentiel. Son organisation hiérarchique correspond à plusieurs niveaux d'index. En effet, un niveau supérieur est un index sur le niveau immédiatement inférieur, lui-même considéré comme un fichier séquentiel. À la construction de l'index, les pages de données sont d'abord allouées séquentiellement et les données sont triées suivant l'attribut d'indexation. Les pages d'index ensuite allouées reçoivent des valeurs discriminantes, qui guident la recherche et des identifiants de pages existantes. Des niveaux supérieurs de pages d'index sont créés jusqu'à un niveau suffisamment faible en volume pour tenir entièrement en mémoire. Des zones de débordement sont également allouées. On notera que seules quelques valeurs de l'attribut d'indexation sont utilisées dans l'index.

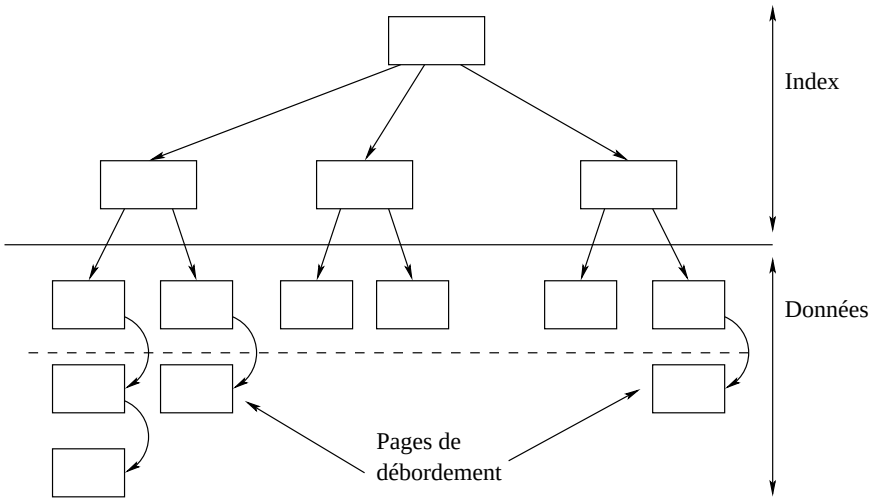


Figure 5.4 – Index ISAM

Un index ISAM (voir figure 5.4) est paramétré par une valeur N qui est le nombre d'entrées dans chaque page d'index de l'arbre. Une entrée d'index est une paire (v, pid) avec v, une valeur discriminante, et pid, un identifiant de page d'index ou de données suivant le niveau dans l'arbre. La zone de données (les feuilles et les pages supplémentaires de la zone de débordement) est indépendante de N. Une page d'index divise un intervalle de la dimension indexée en N intervalles bornés par les valeurs $v_0 < v_1 < \ldots < v_N$. Les valeurs extrêmes du domaine sont implicites : les valeurs v_0 et v_N n'apparaissent donc pas dans la page d'index. Celle-ci contient

N identifiants de page et $N-1$ valeurs discriminantes et a la structure suivante :

$$pid_0; \quad v_1, pid_1; \quad v_2, pid_2; \quad \ldots \quad ; \quad v_{N-1}, pid_{N-1}$$

L'accès à un enregistrement correspondant à une valeur précise nécessite au minimum autant d'accès disque qu'il y a de niveaux de pages d'index. Ce nombre peut être augmenté du nombre de pages de débordement à charger. En effet, les pages de données ne sont plus triées en présence de débordement et doivent donc, dans le pire des cas, être entièrement parcourues. De plus, l'arbre est une structure essentiellement statique car les nœuds internes (les pages d'index) ne sont pas affectés par les modifications et suppressions qui interviennent uniquement au niveau des feuilles. Les entrées d'index restent inchangées et peuvent ainsi utiliser des valeurs discriminantes qui n'existent plus parmi les données. Ce facteur, bien que non pénalisant pour les temps de réponse lors des recherches, peut conduire, en conjonction avec les pages de débordement, à une réorganisation de l'index. La structure de données est remplacée par une structure plus adaptée à l'état courant des données.

5.1.6 Arbres B

À l'instar d'un arbre ISAM, un arbre B est un arbre de recherche général à plusieurs niveaux. Notons qu'il est toujours dense, que les données soient triées ou non. Cette caractéristique est imposée par l'organisation hiérarchique, et l'objectif de répondre aux recherches par égalité sans parcours séquentiel (chaque référence vers les données pointe un enregistrement). Dans le cas où une recherche séquentielle est autorisée, l'index peut être non-dense et les feuilles contiennent alors des références vers des blocs de données.

La proposition initiale de l'arbre B [5] présente une structure de données similaire à l'index ISAM et qui reprend le même principe des valeurs discriminantes et des pointeurs de pages. Cette proposition initiale a connu plusieurs variations [92, 98], dont l'arbre B+ [43], devenu une technique standard des systèmes de gestion de bases de données. La figure 5.5 montre la structure des nœuds internes et des feuilles pour un arbre B+ indexé sur un attribut A. Les différences essentielles avec un arbre ISAM se situent au niveau des feuilles et dans le caractère dynamique de l'arbre.

C'est ainsi que la structure ne définit pas de zone de débordement, mais prévoit l'évolution du volume des données indexées par différents mécanismes qui en font une structure dynamique. Une page de l'index a une capacité maximale de N entrées. Toute page, à l'exception de la racine, voit sa capacité effective k (en nombre d'entrées d'index) respecter la contrainte :

$$\lceil N/2 \rceil \leq k \leq N \ , \tag{5.1}$$

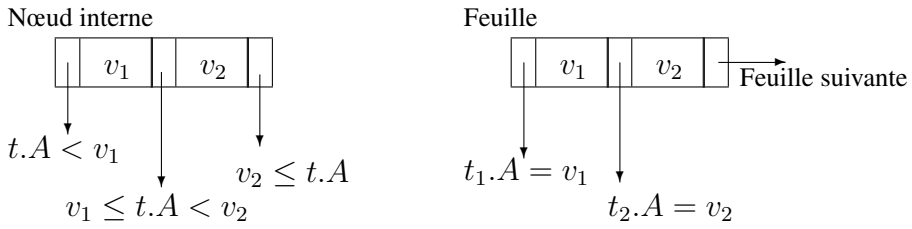


Figure 5.5 – Pointeurs dans un nœud interne et une feuille d'un arbre B+

où $\lceil N/2 \rceil$ désigne le plus petit entier supérieur ou égal à la valeur réelle $N/2$. Ainsi, si $N = 11$, k varie entre 6 ($\lceil 11/2 \rceil$) et 11.

L'arbre est réorganisé par une modification des entrées d'index à chaque allocation de page de données. Une page d'index p_0 de capacité effective $k_0 = N$ devant recevoir une entrée supplémentaire engendre, par partitionnement, deux pages p_1 et p_2 de capacités respectives $k_1 = \lceil N/2 \rceil$ et $k_2 = \lfloor N/2 \rfloor + 1$. La valeur $\lfloor N/2 \rfloor$ désigne le plus grand entier inférieur ou égal à la valeur réelle $N/2$; pour $N = 11$, $\lfloor N/2 \rfloor = 5$. Pour des raisons pratiques, p_1 conserve l'emplacement physique de p_0 et une entrée supplémentaire pour p_2 est rajoutée au parent de p_0 avec la valeur médiane de p_0 comme discriminant. Dans le pire des cas, le partitionnement se poursuit en remontant l'arbre jusqu'à la racine qui donne naissance à une nouvelle racine par son propre partitionnement. Ce cas, par lequel l'arbre grandit en hauteur, intervient de plus en plus rarement (en réalité, de façon logarithmique) au fur et à mesure que l'arbre grandit. Aussi le partitionnement d'un nœud p_0 apporte-t-il à l'arbre une capacité « d'accueil » proportionnelle au nombre de partitionnements en découlant. La réorganisation d'un arbre B+ est donc non seulement locale, mais peu fréquente.

Suivant la contrainte de l'équation (5.1), un nœud de l'arbre (page d'index ou de données) ne peut avoir une capacité inférieure à $\lceil N/2 \rceil$. Lors des suppressions, un nœud dans cette situation déclenche une redistribution de valeurs entre pages de même niveau et de même parent. La redistribution consiste à déplacer une entrée d'index, depuis un nœud frère, vers le nœud en « infraction » vis-à-vis de la contrainte (5.1). Si aucun nœud ne peut se séparer d'une entrée sans enfreindre à son tour la contrainte (5.1), le nœud concerné par la suppression et un nœud voisin fusionnent en un seul nœud. Cette opération peut conduire à une diminution de la hauteur de l'arbre.

Grâce à l'absence de zones de débordement, l'accès à tout enregistrement se fait en un nombre constant d'accès disque car l'arbre est équilibré. Hormis le taux d'occupation, l'arbre B,

en tant qu'index, n'a pas d'inconvénients autres que le besoin en espace de stockage et le surcoût engendré par la maintenance de l'index, éléments inhérents à toute structure d'indexation.

Contrairement à une structure ISAM, les feuilles d'un arbre B (et de ses variations, y compris l'arbre B+) sont chaînées. Un parcours séquentiel est donc possible : il suffit d'utiliser l'index pour atteindre le premier élément de la séquence. Les recherches par intervalles sont ainsi plus aisées qu'avec un index ISAM. Cependant, les recherches par égalité sont moins efficaces qu'avec une table de hachage.

En raison de ses avantages, l'arbre B+ est la méthode d'accès monodimensionnelle par excellence, implémentée dans tous les SGBD. En pratique, la valeur N est choisie de manière à être un nombre pair ; la valeur entière $d = N/2$ est communément appelée la dimension de l'arbre.

5.2 Index multidimensionnels

Il est généralement admis qu'un index multidimensionnel est préférable à plusieurs index monodimensionnels dans un contexte de requêtes multi-critères. En effet, un système dépourvu d'index multidimensionnels doit assurer la maintenance d'un certain nombre de structures d'indexation. En supposant que tous les attributs sont indexés, utiliser simultanément plusieurs index monodimensionnels nécessite de réaliser une intersection des ensembles d'enregistrements fournis par chaque index. En outre, on montre [93] qu'une recherche dans ce contexte est inefficace car les sélectivités individuelles des structures de données ne peuvent pas être combinées pour réduire rapidement l'espace de recherche.

Les index multidimensionnels apportent une solution à ces inconvénients : une unique structure de données, aucune opération d'intersection entre des ensembles parfois importants, et une sélectivité nettement meilleure dans le traitement des requêtes multi-attributs. Cependant, cette option, qui consiste à prendre en compte plusieurs dimensions à la fois lors des recherches, présente des difficultés dues à la nature des espaces représentés. Le caractère non-trivial des particularités des espaces à d dimensions a donné lieu à de nombreux travaux qui seront abordés dans la suite de cette section. Ces particularités sont responsables de l'inefficacité des techniques d'indexation lorsque le nombre de dimensions croît.

D'une manière générale, les techniques d'indexation sur d dimensions procèdent à un partitionnement de l'espace en sous-espaces (ou volumes) dénommés « cellules ». Une cellule contient a priori plusieurs enregistrements. Elle est donc de granularité supérieure par rapport aux enregistrements. Ceci permet de réduire le volume des informations nécessaires à l'abou-

tissement d'une recherche. Cette réduction n'est malheureusement pas suffisante pour contrebalancer le phénomène, connu sous le nom de « malédiction de la dimensionnalité », de dégradation des performances en raison de la croissance exponentielle du nombre de cellules.

Considérons le cas d'un espace E de dimension d divisé sur une dimension en n sous-espaces de dimension d par des hyperplans de dimension $d-1$, éventuellement parallèles entre eux. La répétition de cette subdivision sur chaque dimension conduit à n^d cellules. Le nombre de cellules est donc clairement exponentiel par rapport à la dimension, et linéaire par rapport au nombre de subdivisions. Lorsque le paramètre d est grand, il est probable que le nombre de cellules avoisine ou dépasse le nombre d'objets (enregistrements) à indexer. Le bénéfice apporté par une réduction du volume d'informations diminue alors et il peut être plus coûteux d'utiliser l'index que de parcourir tout l'espace en examinant chaque objet.

Dans la suite de cette section, sont présentées les méthodes d'indexation qui prennent en compte plusieurs dimensions. En premier lieu, seront présentées des structures de données dont l'intérêt est de rassembler les caractéristiques que l'on retrouve dans les index. Ces structures de données pourraient elles-mêmes servir d'index, mais elles seraient d'une faible efficacité. Dans un second temps, quelques techniques multidimensionnelles, majoritairement arborescentes, seront traitées. Certaines d'entre elles s'inspirent des tables de hachage et les étendent de manière à gérer plusieurs dimensions.

5.2.1 Structures de données

5.2.1.1 K-D-Tree

Cette structure de données proposée par Bentley [8] est la structure de base de tous les index arborescents. Elle est formée d'un arbre de recherche binaire dont chaque nœud reflète un partitionnement de l'espace E à k dimensions (d'où le nom de la structure) en deux sous-espaces E_1 et E_2 de dimension k. Le partitionnement est réalisé grâce à un hyperplan de dimension $k-1$: une droite dans le plan, un plan dans un espace à 3 dimensions, etc. Les hyperplans sont iso-orientés, c'est-à-dire parallèles à un axe de référence de E : dans un espace cubique, ils seraient parallèles à l'une des faces du cube.

La stratégie de subdivision de l'espace impose une alternance cyclique des axes de référence ; en dimension 2, la séquence des axes serait (x, y, x, y, x, ...) ou (y, x, y, x, y, ...). Elle impose également que le plan de partitionnement passe par un objet de l'espace car cet objet est stocké dans un nœud. Les branches du nœud représentent les deux sous-espaces issus du partitionnement, lequel est répété jusqu'à ce que tous les objets soient représentés dans l'arbre. La

structure ne gérant que des points, les nœuds internes et les feuilles sont des points. À l'image des arbres B, le k-d-tree (pour « k-dimension tree ») présente l'inconvénient de stocker des données au sein des nœuds. Il est également sensible à l'ordre (voir section 5.2.3.1). L'alternance cyclique des dimensions peut être un facteur d'inefficacité car elle conduit à des arbres déséquilibrés. La figure 5.6 montre un exemple d'arbre k-d.

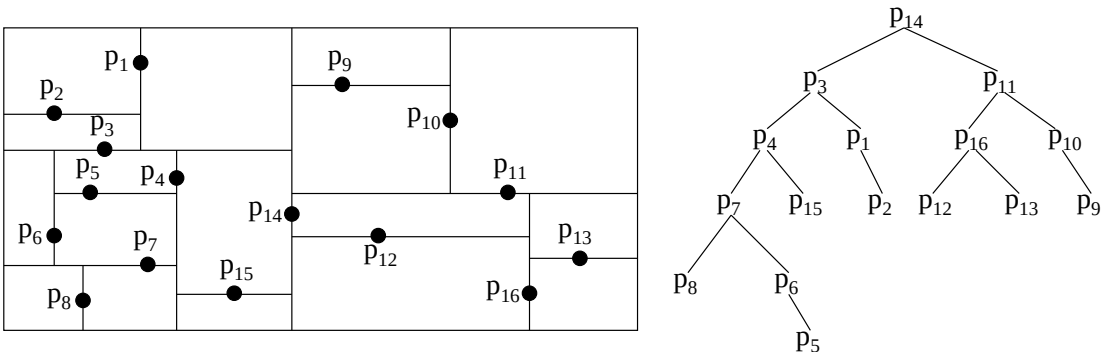


Figure 5.6 – K-D-Tree

Ces inconvénients sont traités par une amélioration du k-d-tree, l'*adaptive k-d-tree* [9]. Dans cette structure, l'hyperplan de partitionnement est choisi de manière à équilibrer le nombre de points dans les sous-espaces résultants. L'alternance des axes de partitionnement n'est plus une règle. Pour pallier l'absence d'alternance implicite, les informations stockées dans un nœud interne décrivent l'hyperplan de subdivision. En conséquence, tous les points sont référencés au niveau des feuilles. La condition d'arrêt du partitionnement est atteinte lorsque le nombre de points dans les feuilles passe en dessous d'un certain seuil. La valeur de ce seuil est habituellement fixée par l'espace alloué à la cellule sur le support de stockage. La structure obtenue, qui reste sensible à l'ordre, est plus importante en volume en raison des informations sur les plans de séparation. L'exemple de la figure 5.7 est équilibré parce que tous les points sont connus a priori. Dans le cas où des mises-à-jour surviennent, la structure ne conserve pas cette propriété.

Gaede et Günther [68] considèrent le bintree de Tamminen [134] comme une autre variante du k-d-tree. Les hyperplans de partitionnement sont implicites car la subdivision produit des sous-espaces de même taille.

5.2.1.2 BSP-Tree

Dans ses principes, le Binary Space Partition Tree [63] est similaire à l'adaptive k-d-tree ; l'espace à indexer est subdivisé récursivement en sous-espaces de dimensions k au moyen d'hy-

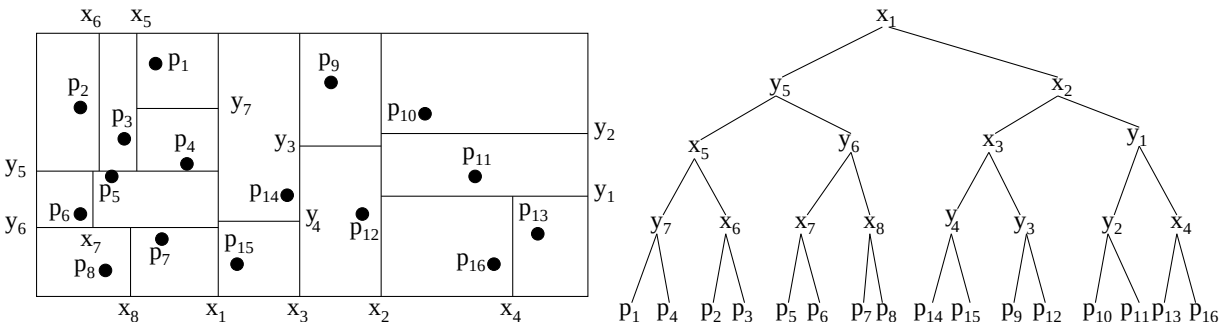


Figure 5.7 – Adaptive k-d-tree

perplans de dimensions $k-1$ jusqu'à ce que le nombre d'objets soit en dessous d'un seuil fixé. Cependant, il s'en distingue par une différence de poids : les hyperplans ne sont plus nécessairement iso-orientés. Ceci permet d'adapter le partitionnement à la répartition des objets dans l'espace, puisque c'est cette répartition qui détermine l'emplacement de l'hyperplan. Le BSP-Tree permet d'indexer des objets spatiaux (dotés d'une aire ou d'un volume) en plus des points.

Cette structure, bien que très utilisée dans le domaine du graphisme [62, 145], notamment pour l'élimination des surfaces cachées, souffre de quelques handicaps. La détermination des hyperplans et les calculs d'intersection sont coûteux. De même, l'espace de stockage requis est plus important : il faut en effet conserver les informations sur l'hyperplan (emplacement et orientation) alors qu'auparavant, une position relative à l'axe concerné suffisait. L'arbre obtenu n'est pas équilibré et peut avoir une grande profondeur. La figure 5.8 montre un exemple de BSP-Tree sur le même ensemble de points qu'en figure 5.7.

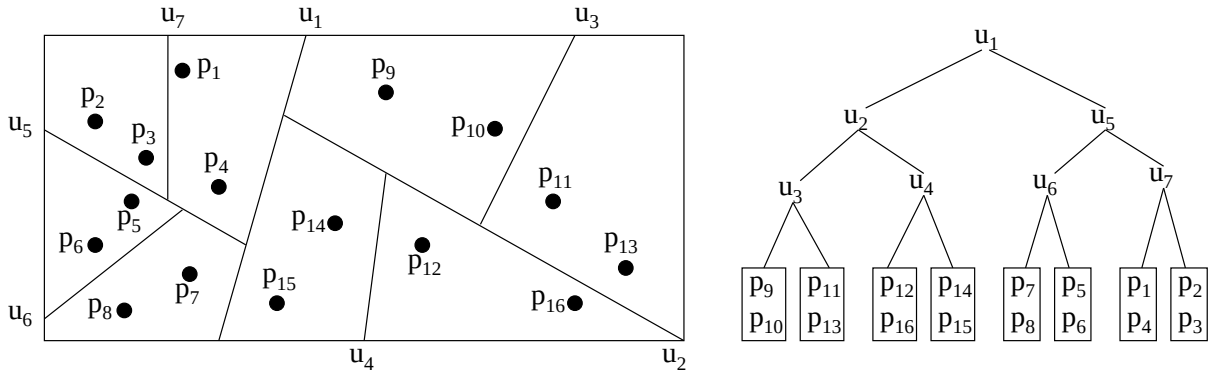


Figure 5.8 – BSP-Tree

5.2.2 Techniques d'indexation

La littérature sur les index multidimensionnels est riche en recherches sur les techniques d'indexation [4, 6, 12, 13, 30, 33, 37, 76, 80, 81, 89, 100, 101, 119, 129, 130, 146, 148]. Mais l'intégration concrète de ces techniques (ou « méthodes d'accès ») dans un système est marginale : en dehors du R-Tree de Guttman [76] (discuté en section 5.2.2.1), et plus rarement du R*-Tree de Beckmann *et al.* [6] (section 5.2.2.1), seul le Z-ordering [108] est, à notre connaissance, disponible dans un SGBD, en l'occurrence, Oracle.

Toutes les techniques d'indexation, basées soit sur les structures de données de la section 5.2.1 (voir [73] par exemple), soit sur une technique existante, présentent néanmoins des idées originales. Les structures de données (de la section 5.2.1) déterminent notamment la politique de partitionnement adoptée par la technique : les index basés sur le k-d-Tree partitionnent l'espace à indexer de façon régulière tandis que ceux basés sur le BSP-Tree effectuent un partitionnement dicté par le placement des données dans l'espace.

Les propositions d'index étant nombreuses, cette section ne présentera que les techniques (assimilées à leur structure de données) qui se distinguent notablement des autres, en raison d'une spécificité unique ou de leurs performances.

5.2.2.1 R-Tree

La technique d'indexation du R-Tree, proposée par Guttman en 1984 [76] est, sur le plan historique, la première technique d'indexation multidimensionnelle. Elle a été proposée pour prendre en charge les objets spatiaux dans un espace multidimensionnel. Un objet est spatial quand il a un volume (une surface dans un espace à 2 dimensions) non-nul. Il est défini par la donnée d'un nombre de points supérieur à la dimension de l'espace auquel il appartient. On trouve ces objets dans les applications géographiques ou la conception assistée par ordinateur, où les objets manipulés sont des polygones plutôt que des points.

Dans un R-Tree, un objet d'un espace E de dimension d est représenté par un intervalle $I_i = [a_i, b_i]$ sur chaque dimension i, l'intervalle étant le plus restreint possible pour contenir la projection de l'objet sur l'axe de la dimension i. Notons cependant que l'intervalle peut être ouvert car la technique autorise des bornes infinies. L'hyperrectangle constitué par les intervalles $I_1 = [a_1, b_1], I_2 = [a_2, b_2], \ldots, I_d = [a_d, b_d]$ décrit la région englobante minimale (ou REM) de l'objet Ω considéré par un intervalle multidimensionnel I : $I(\Omega) = I_1(\Omega) \times I_2(\Omega) \times \ldots \times I_d(\Omega)$. La structure de données en arbre du R-Tree figure une hiérarchie de régions (ici des rectangles) englobantes minimales. Il est possible que des portions de l'espace soient couvertes par plusieurs REM ; on parle alors d'*overlap* (voir section 5.2.3.3). Les objets contenus dans ces ré-

gions font l'objet de plusieurs entrées d'index (voir section 5.1) au niveau des feuilles. En conséquence, la recherche de l'un de ces objets suivra plusieurs chemins distincts dans l'arbre. Dans la suite de ce document, nous utiliserons l'acronyme « REM » pour désigner aussi bien une région englobante minimale pour le cas général, qu'un rectangle englobant minimal pour le cas le plus fréquent.

Le fonctionnement de l'arbre est similaire à celui d'un arbre B. Une entrée d'index est également constituée d'informations (un REM) sur les données indexées et d'un pointeur vers une page d'index. Les feuilles contiennent, outre des REM, des pointeurs vers la description des objets spatiaux ou des points. Tous les nœuds, exception faite de la racine, ont un nombre minimal m d'entrées et une capacité effective k telle que $m \leq k \leq N$, N étant la capacité maximale d'un nœud, déterminée par la taille de la page allouée sur le support physique pour chaque nœud. Lorsque la capacité maximale est atteinte lors d'une insertion, le nœud est partitionné et remplacé par deux nœuds au même niveau. Chaque nœud correspond à une région choisie de manière à minimiser le volume total des REM. L'arbre étant équilibré, toutes les feuilles sont à une même hauteur h inférieure à $\lceil \log_m(N) \rceil$.

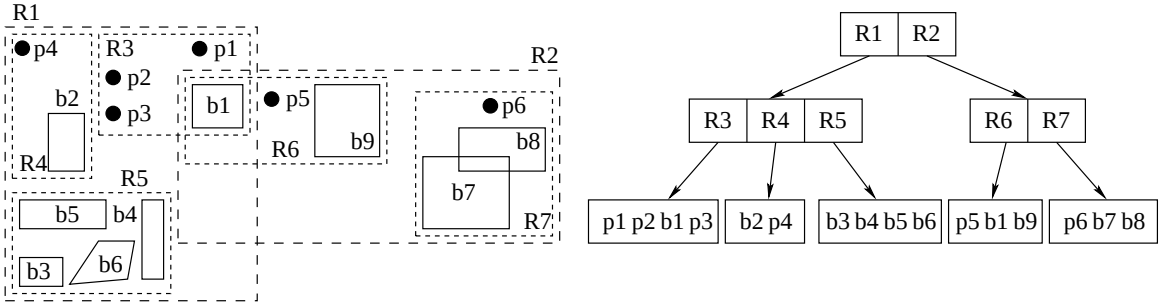


Figure 5.9 – R-Tree

La recherche d'un objet (par exemple b1 dans la figure 5.9) se fait en spécifiant un intervalle I_s de dimension d. Dans un nœud, l'intersection de I_s avec chaque REM d'un sous-nœud est calculée. Une intersection non-vide indique que le nœud sera visité. Toutes les entrées sont testées car la possibilité d'un recouvrement dénote implicitement plusieurs (et non un seul) rectangles englobants, en l'occurrence, R3 et R6 pour l'objet b1. Gaede et Günther [68] pointent le fait qu'un R-Tree ne peut pas réellement effectuer une recherche exacte à moins que l'objet ne soit constitué que de facettes parallèles aux axes (ce qui est le cas pour b1). Pour un tel objet, l'hyperrectangle englobant occupe le même volume que l'objet. Dans les autres cas d'objets non iso-orientés comme b6, la recherche sélectionne des « objets candidats », c'est-à-dire ceux

dont le rectangle englobant intersecte I_s. Par la suite, l'objet lui-même est testé et sa validité en tant que résultat est déterminée.

Lors de la suppression d'un objet, la méthode d'accès mène une recherche pour localiser la feuille correspondant à l'objet. Les données sur l'objet (REM et référence) sont supprimées de la feuille. Quelle que soit l'évolution de la capacité effective de la feuille, un ajustement du REM, c'est-à-dire une contraction d'un ou plusieurs intervalles, survient pour chaque parent jusqu'à la racine. Si $k < m$, les entrées d'index sont soit redistribuées aux nœuds frères, soit réinsérées dans l'arbre. Un nouvel ajustement propagé vers le haut peut être nécessaire pour finaliser la suppression.

En tant que première méthode d'indexation multidimensionnelle, qui plus est, capable de gérer des objets spatiaux, le R-Tree a donné lieu à de nombreuses variantes. Le « packed R-Tree » [122] par exemple, est capable de déterminer le meilleur partitionnement de l'espace et l'arbre minimal qui lui correspond, à condition que tous les objets soient connus. Le « sphere Tree » [139], autre variante du R-Tree, s'en distingue en utilisant des sphères comme volumes englobants au lieu de rectangles.

R*-Tree Le R*-Tree de Beckmann, Kriegel, Schneider et Seeger [6] est une variante du R-Tree dont l'objectif est d'en corriger certaines faiblesses. En particulier, les auteurs ont constaté que la phase d'insertion des objets dans l'arbre influençait grandement les performances de la structure lors des recherches. Contrairement au R-Tree où un nœud dont la capacité maximale est dépassée subit systématiquement un partitionnement, cette méthode réinsère une proportion p des entrées du nœud dans la racine de l'arbre avant de se résoudre à partitionner le nœud plein. La proportion p d'entrées réinsérées dans l'arbre afin d'éviter un nouveau partitionnement est un paramètre de la méthode. Cependant, la réinsertion peut quand même conduire à une modification structurelle de l'arbre.

Les objectifs visés lors du partitionnement des nœuds pleins sont également différents :

- minimiser les recouvrements entre REM ;
- minimiser le périmètre des rectangles ;
- maximiser la capacité effective des nœuds.

Au final, les rectangles englobants sont prioritairement disjoints et de forme plus carrée. La figure 5.10 montre un exemple de R*-Tree.

Pour les opérations autres que l'insertion, les algorithmes restent les mêmes, mais les performances globales de la structure sont meilleures car la recherche est globalement plus efficace.

Cette technique est parfois considérée comme la meilleure structure d'index pour les données de faible dimension, notamment bidimensionnelles [20, 100].

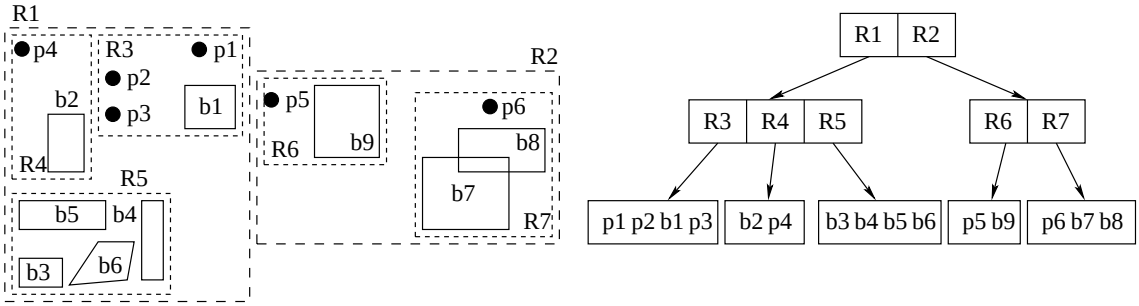


Figure 5.10 – R*-Tree

X-Tree La technique du X-Tree est une variante du R*-Tree spécifiquement conçue pour les grandes dimensions (typiquement, supérieures à 10). Cette structure, proposée par Berchtold, Keim et Kriegel [13], vise le principal problème des grandes dimensions : le recouvrement (voir la section 5.2.3.3). Elle introduit deux notions originales, le *super-nœud* et l'historique des partitionnements.

Les travaux effectués sur l'indexation de données ont permis de tirer des conclusions quant à l'organisation d'index la plus adaptée suivant la nature des données. Ces conclusions, qui seront exploitées par l'X-Tree, montrent qu'une hiérarchie est l'organisation la plus efficace pour les données de faible dimension (par exemple, l'arbre B+ en monodimensionnel et le R*-Tree pour 2 à 6 dimensions). À l'inverse, une organisation linéaire (sous forme de tableau) est plus efficace en moyenne pour des données de grande dimension. C'est typiquement le cas du parcours séquentiel. Une hiérarchie peut être efficace sur de grandes dimensions, à la condition que les recouvrements entre les unités d'indexation (c'est-à-dire les cellules) soient inexistants. Une telle condition est insatisfaisable dès lors que des objets spatiaux sont représentés par des volumes englobants. On peut dériver une organisation linéaire d'une organisation hiérarchique en limitant l'arbre à ses feuilles, les niveaux supérieurs étant ignorés.

Un nœud qui a atteint sa capacité maximale et auquel est affectée une entrée d'index supplémentaire doit être partitionné. Mais cette opération a tendance à engendrer des recouvrements de REM. Dans un X-Tree, l'algorithme d'insertion essaie de partitionner le nœud en limitant le pourcentage de recouvrement produit en dessous d'un taux paramétrable T_x. Idéalement, $T_x = 0$. Si une telle possibilité n'existe pas, l'opération de partitionnement est reportée et le nœud est promu au rang de super-nœud. Le super-nœud est différent d'un nœud « normal » en

ce sens qu'il a dépassé la capacité nominale. Autrement dit, ne sont autorisés que les partitionnements réalisables sans recouvrement.

La deuxième notion propre à l'X-Tree est l'historique des partitionnements. L'historique est utilisé lors des opérations de partitionnement afin de trouver la dimension qui permettra de ne produire aucun recouvrement. Cette dimension est une de celles suivant lesquelles tous les rectangles englobants des nœuds fils ont auparavant été partitionnés. L'historique est stocké sous la forme d'un arbre binaire dont les feuilles sont les REM présents dans l'arbre d'index. Une opération de partitionnement étant toujours un remplacement d'un REM par deux REM, il suffit d'étiqueter les nœuds internes de l'arbre binaire avec la dimension de partitionnement pour retracer entièrement l'historique.

En règle générale, les nœuds de l'arbre tendent à se transformer en super-nœuds lorsque la dimension augmente. La figure 5.11, tirée de [13], montre l'évolution de la forme d'un X-Tree telle que les auteurs l'ont constatée expérimentalement. Un rectangle plein représente un super-nœud tandis qu'un rectangle vide représente un nœud « normal ».

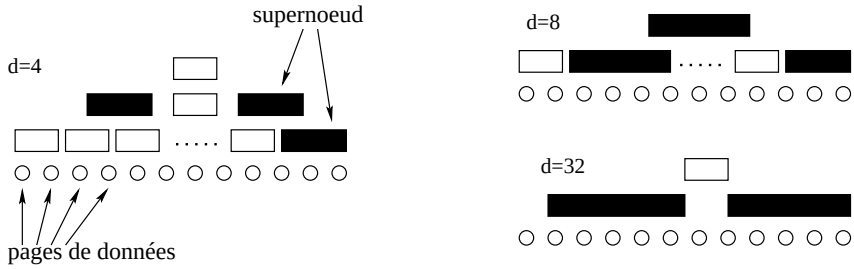


Figure 5.11 – Forme générale d'un X-Tree

5.2.2.2 Pyramid-Tree

Le Pyramid-Tree [12] est une autre proposition de Berchtold, avec Böhm et Kriegel, postérieure à l'X-Tree. Cet index est capable de performances supérieures à celles d'un X-Tree sur les mêmes données d'un facteur allant jusqu'à 14 en ce qui concerne les accès disque et jusqu'à 2500 pour les temps de réponse.

De telles performances sont possibles car la technique n'indexe pas d'objets spatiaux : elle échappe donc au problème des recouvrements des régions englobantes. Les performances se justifient également par le fait que la technique ne subit pas la malédiction de la dimensionnalité, c'est-à-dire la dégradation de ses performances lorsque la dimension augmente. Pour réussir cette prouesse, un partitionnement original « optimisé pour les grandes dimensions »

est utilisé : un hyperrectangle de dimension d est divisé en un nombre $N = 2.d$ de pyramides dont le sommet est le centre de l'hyperrectangle. La figure 5.12-a montre un exemple pour des hyperrectangles de dimensions 2 et 3.

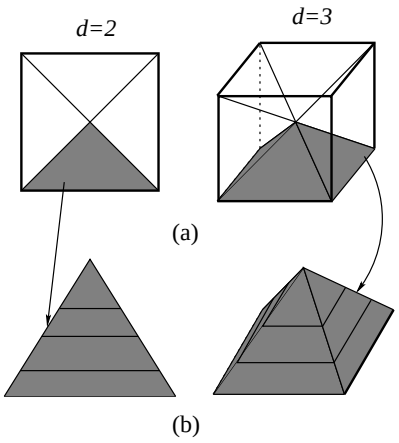


Figure 5.12 – Partitionnement en pyramides

Chaque pyramide est divisée en tranches, parallèles à la base, associées chacune à une page de données (figure 5.12-b). Un identifiant unique est assigné à chaque pyramide. Un point de l'espace est représenté par un nombre réel combinant l'identifiant de la pyramide à laquelle il appartient et le numéro de la tranche. Ainsi, la technique procède à une transformation de points de dimension d en points de dimension 1. Dès lors, l'indexation de l'espace multidimensionnel se réduit à une indexation classique, en l'occurrence, à l'aide d'un arbre B+.

La transformation utilisée à l'insertion d'un point intervient également pour les recherches aussi bien exactes que par intervalle. En raison de la représentation dans un espace de dimension inférieure, les collisions déjà évoquées dans les tables de hachage (voir section 5.1.4) sont inhérentes à la technique. La recherche nécessite donc un filtrage des points sélectionnés afin d'exclure les faux positifs dûs aux collisions.

D'autres méthodes (par exemple, le Z-ordering [108] ou celles qui effectuent un hachage) sont basées sur une transformation vers un espace de dimension inférieure. Mais le Pyramid-Tree est la seule méthode qui reste réellement efficace dans les grandes dimensions. Malgré ses performances, elle n'a pas que des avantages. Cette technique est limitée aux points, ce qui l'exclut du champ d'application des domaines géographiques et géométriques qui ont motivé les premières méthodes multidimensionnelles d'indexation. De plus, ses performances sont optimales seulement pour des données uniformément distribuées. Bien que Berchtold *et al.* pro-

posent dans [12] un remplacement du centre géométrique par le centre de gravité, le problème d'équilibre resterait ouvert d'après Berrani *et al.* [14].

5.2.2.3 TV-Tree

Le TV-Tree (pour *Telescopic Vector Tree*) de Lin, Jagadish et Faloutsos [100] est une structure dont l'objectif de conception est de repousser la dégradation inévitable des performances vers des dimensions supérieures. Pour ce faire, et puisque la dégradation est exponentielle en fonction de la dimension, la structure prend en compte un nombre de dimensions plus faible à la racine que dans les feuilles. Ainsi, l'impact de la dimensionnalité est plus faible dans les niveaux supérieurs, proches de la racine. En conséquence, l'arbre est plus petit en volume et le vecteur représentant les valeurs prises en compte pour une recherche varie en taille avec le niveau dans l'arbre, d'où le nom de la technique.

À chaque niveau, le TV-Tree divise les attributs des données du sous-arbre en :

- attributs de valeurs communes aux données indexées, déjà utilisés plus haut dans l'arbre ;
- attributs ignorés, dont la prise en compte augmenterait l'impact de la dimensionnalité ;
- attributs discriminants.

Mais des informations préalables sur les attributs sont nécessaires pour choisir efficacement les attributs discriminants à chaque niveau.

D'après les auteurs, cette façon de discriminer, en utilisant de plus en plus de critères, correspond à la façon dont les humains classent intuitivement des objets, par exemple en zoologie où les espèces sont distinguées en vertébrés et invertébrés. La distinction entre sang chaud et sang froid n'est valable que pour les vertébrés. Par conséquent, cet attribut ne sera utilisé que dans le sous-arbre des vertébrés.

5.2.2.4 Hybrid Tree

L'Hybrid-Tree de Chakrabarti et Mehrotra [33] est une hybridation de techniques arborescentes existantes afin d'en mutualiser les avantages. Les auteurs distinguent parmi les techniques d'indexation multidimensionnelle celles qui utilisent des régions englobantes (par exemple, le R-Tree, section 5.2.2.1) et celles qui partitionnent l'espace en cellules disjointes (par exemple, le kDB-Tree [119]). Les premières souffrent du recouvrement des REM, et les secondes ne peuvent garantir un taux minimal d'utilisation des pages physiques allouées (voir section 5.2.3.7).

Le problème du recouvrement des REM a déjà été abordé pour le R*-Tree et l'X-Tree en sections 5.2.2.1. Le taux minimal d'utilisation désigne la proportion plancher du nombre d'en-

trées dans une page d'index par rapport à la capacité disponible, toutes les techniques faisant correspondre une page physique à un nœud de l'arbre. En général, les nœuds sont partitionnés lorsqu'ils atteignent leur capacité maximale. On obtient alors deux nœuds distincts qui remplacent le nœud précédent, comme c'est le cas dans un arbre B (section 5.1.6). Lorsque le partitionnement du nœud n'est pas restreint à une région de l'arbre, il provoque généralement une cascade de partitionnements forcés, y compris sur les nœuds dont la capacité maximale n'est pas atteinte. Les pages résultantes sont alors largement sous-occupées. En conséquence, l'arbre d'indexation est grand (en nombre de nœuds) et volumineux (en taille occupée), et les accès disque sont très peu efficaces car les nœuds lus apportent peu d'informations.

La technique du Hybrid-Tree essaie de mitiger l'impact de ces deux problèmes : les recouvrements ne sont admis que si l'option contraire déclenchait une cascade de partitionnements. Pour cette structure, une politique de partitionnement peu courante est utilisée : le partitionnement est monodimensionnel. Ainsi, les différents fils d'un nœud ne se différencient que sur une dimension, ce qui a pour effet d'augmenter la sélectivité des nœuds au cours des recherches (voir la section 5.2.3.6). Le choix de la dimension de partitionnement est à la charge d'algorithmes, proposés dans la même publication [33], qui se fondent sur des probabilités pour optimiser les futures opérations de recherche.

5.2.2.5 Grid File

Le Grid File est une structure d'indexation non-hiérarchique proposée par Nievergelt, Hinterberger et Sevcik [105], qui constitue une extension du hachage indirect (voir section 5.1.4) à plusieurs dimensions.

Le principe est toujours de subdiviser l'espace de données en cellules, ce qui revient, dans l'esprit de la technique, à superposer une grille sur l'espace. La grille est figurée par les pointillés dans la figure 5.13. Les cellules obtenues, de tailles diverses, disposent chacune d'une entrée dans un répertoire (ou « grid directory »). De même que le hachage monodimensionnel associe un emplacement à une valeur de la fonction de hachage, chaque cellule multidimensionnelle est associée à un emplacement (ou « data bucket ») de l'espace (linéaire) de stockage physique. L'emplacement correspond à une page physique de données, parfois partagée entre plusieurs cellules. La fonction de hachage prend en compte plusieurs dimensions pour calculer l'entrée du répertoire où trouver l'objet recherché.

L'écartement entre les hyperplans qui partitionnent une dimension est variable. Il est généralement fixé par la capacité maximale d'un emplacement. La relation hiérarchique qui existe dans les index arborescents est ici inexistante. Ceci limite le nombre d'accès disque nécessaire

pour toute opération dans l'index. En réalité, toute recherche exacte aboutit en un maximum de deux accès disque : un accès pour lire l'entrée du répertoire calculée (si l'entrée n'est pas disponible en mémoire), et un accès pour charger la page de données désignée par le répertoire.

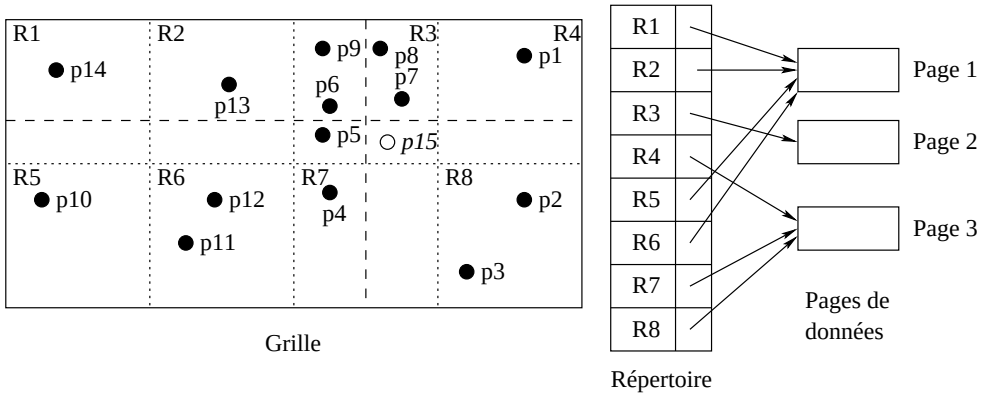


Figure 5.13 – Grid File

Cependant, cette technique admet des redondances de pointeurs qui conduisent à une croissance linéaire du répertoire. Par exemple, dans la figure 5.13, la page 1 est référencée plusieurs fois, de même que la page 3. En outre, l'insertion d'un seul point dans l'espace de données peut générer une forte croissance du répertoire.

Pour une opération d'insertion, l'entrée adéquate est calculée par la fonction de hachage. Si l'emplacement désigné n'a pas atteint sa capacité maximale, l'insertion ne modifie pas le répertoire. Par contre, si l'emplacement est plein (par exemple, à l'insertion de p15 dans R3 pour une capacité de cinq entrées), une page de données est allouée et deux options existent. Dans le cas où plusieurs cellules pointent le même emplacement et qu'un hyperplan existant les partitionne déjà, des points sont déplacés vers la nouvelle page et les entrées sont mises à jour. Autrement, un nouvel hyperplan (figuré par des tirets dans la figure 5.13) divise la cellule concernée, ce qui se répercute sur toutes les autres cellules intersectées. Dans notre exemple, si le plan vertical est utilisé, la cellule R7 est partitionnée. Si l'hyperplan horizontal est utilisé, les cellules R1, R2 et R4 sont partitionnées et produisent des cellules vides. La fonction de hachage est modifiée en conséquence du nouvel hyperplan et un éventuel ajustement des pointeurs permet de regrouper des cellules sous-occupées. Il n'en reste pas moins que une ou trois entrées sans utilité immédiate sont rajoutées au répertoire.

La famille des techniques d'indexation par hachage comprend d'autres techniques qui sont des variantes du Grid File, et dont l'objectif est de ralentir la croissance du répertoire. Par exemple, Hinrichs [82] propose de rajouter un deuxième répertoire (« root directory ») au-

dessus du répertoire « normal », ce qui permet de confiner certains partitionnements. Une autre technique, le *twin grid file* par Hutflesz, Six et Widmayer [85], utilise deux grilles indépendantes sans relation hiérarchique, c'est-à-dire à un même niveau. La grille la plus appropriée est utilisée suivant la distribution des données : les points sont déplacés d'un emplacement pointé par une grille vers l'autre grille si le transfert permet de limiter l'augmentation du nombre de cellules ou de faire baisser ce nombre.

BANG File Cette technique d'indexation, par Michael Freeston [60], est une technique hybride basée sur un hachage multidimensionnel, mais qui utilise un arbre pour représenter le répertoire.

Comme dans le Grid File, l'espace de données est divisé par une grille, mais les cellules obtenues peuvent présenter un recouvrement. La figure 5.14 montre un exemple pour une capacité de 4 entrées par page de données. Le recouvrement est possible grâce à une imbrication des cellules : une cellule de la grille initiale en dépassement de capacité (R3 dans la figure 5.14 à l'ajout de p9) est bien sûr partitionnée par un hyperplan. Cependant, au lieu de diviser toutes les cellules intersectant l'hyperplan, seule la cellule concernée (R3) est partitionnée. L'organisation du répertoire en arbre est apte à représenter le caractère local du partitionnement. En conséquence, une seule entrée est rajoutée au répertoire (celle de R4).

Le recouvrement observé ici ne souffre pas de l'inconvénient constaté dans les techniques purement arborescentes car il ne correspond pas à une intersection, mais à une imbrication. Un objet dans un recouvrement (p5 par exemple) n'a qu'une entrée dans l'arbre. Le BANG (Balanced And Nested Grid) File est dit équilibré car le nombre d'objets représentés est à peu près équivalent dans les branches droite et gauche d'un nœud de l'arbre des grilles.

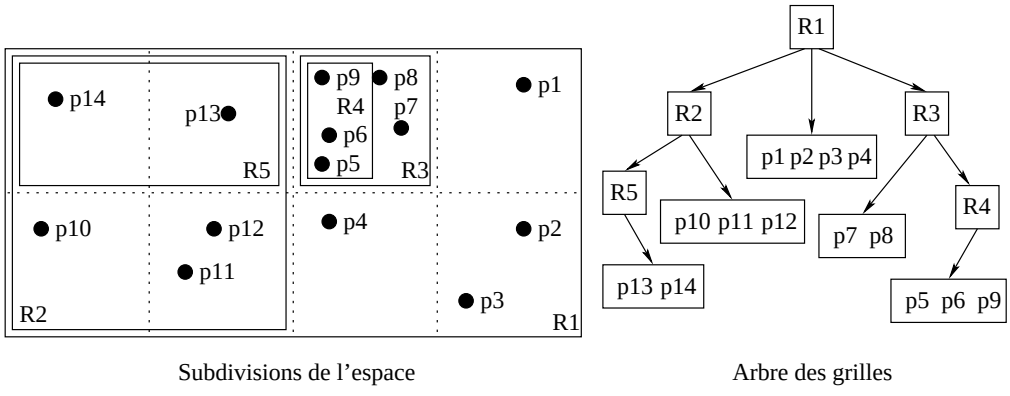


Figure 5.14 – BANG File

Notons toutefois que les régions représentées par certains emplacements ne sont plus nécessairement des régions régulières : dans la figure 5.14, R1 n'est pas réellement un rectangle puisque la région représentée ne comprend ni R3 ni R2.

5.2.3 Propriétés des techniques d'indexation

Les diverses techniques d'indexation multidimensionnelle disponibles sont considérées dans la littérature (notamment dans [14, 20, 68, 155]) sous plusieurs aspects dont l'évaluation donne une idée de leur performance. Il s'agit notamment des propriétés mathématiques des concepts et des structures de données, en général dictées par les objectifs de conception. Ces propriétés influent sur les propriétés pratiques de mise en œuvre des techniques. Les autres aspects portent sur les caractéristiques de l'arbre construit car, à quelques exceptions près, les techniques d'indexation multidimensionnelle créent des index arborescents. Cette section passe en revue les propriétés recensées.

5.2.3.1 Sensibilité à l'ordre

La structure de données obtenue en résultat de l'indexation d'un jeu de données peut être différente suivant l'ordre de prise en compte des données. Cette différence est fonction des critères de placement et de la politique de partitionnement que définit la technique. Si une telle différence d'organisation de la structure existe, la technique est dite sensible à l'ordre.

La sensibilité à l'ordre (parfois désignée par le terme « indéterminisme ») implique qu'aucune garantie ne peut être donnée que la structure obtenue (en général un arbre) est optimale. En effet, il pourrait exister une autre forme de la structure, induite par un ordre différent d'insertion des données, plus « adaptée » d'après d'autres critères de performance comme la complexité spatiale (section 5.2.3.9), le taux d'occupation (section 5.2.3.7) ou l'équilibre de l'arbre (section 5.2.3.10). C'est ainsi que le R*-Tree [6] améliore le R-Tree [76] en ciblant spécifiquement ce problème d'indéterminisme.

5.2.3.2 Forme des régions

La modélisation géométrique des données structurées associe un domaine d'attribut à une dimension d'un espace multidimensionnel où un enregistrement correspond à un point de l'espace. Les coordonnées du point dans l'espace sont données par les valeurs de l'enregistrement correspondant.

Comme indiqué en section 5.2, les cellules utilisées dans l'index regroupent les représentations géométriques des enregistrements. Elles définissent des régions de l'espace multidimensionnel englobant un nombre variable de points. La région ou forme englobante est le plus souvent un hyperrectangle, en raison de la relative simplicité de ce dernier concept, ou une hypersphère. D'autres formes plus complexes mais moins intuitives existent ([12, 89]), par exemple, résultant de l'intersection d'un hypercube et d'une hypersphère.

5.2.3.3 Recouvrement

La notion de recouvrement (*overlap*) désigne l'intersection entre les cellules de l'espace multidimensionnel. D'après Berchtold *et al.* dans [13], cette caractéristique est un problème important au point de justifier une méthode d'indexation (celle du X-Tree, traitée en section 5.2.2.1) qui vise à le résoudre spécifiquement. Il serait également à l'origine de la dégradation des performances : la proportion d'objets couverts par plus d'une cellule (taux de recouvrement) augmente très rapidement pour atteindre 80% en dimension 4 et « approcher » 100% pour les dimensions supérieures à 6 sur des données réelles. On note que le R^+-Tree [130] vise également à résoudre ce problème que rencontre le R-Tree [76].

L'existence de recouvrements exprime la possibilité d'atteindre un objet par plusieurs chemins différents. L'impact de ce taux sur les performances des techniques se justifie par le fait que les objets faisant partie du résultat final sont « extraits » de plusieurs cellules. Toutes ces cellules sont donc examinées au cours de la recherche, y compris celles, superflues, dont les objets font déjà partie du résultat. Un taux de recouvrement plus faible ou nul aurait permis de faire l'économie du parcours de toutes ces cellules, sachant qu'un nœud correspond à une ou plusieurs pages sur le support de stockage.

5.2.3.4 Réduction de la dimension

La réduction de la dimension d'un espace est une tentative de contournement des problèmes liés aux espaces de grande dimension et évoqués en section 5.2 (voir aussi [14, 147]). Cette idée découle du constat que les données réelles sont fortement corrélées et groupées. Il n'est donc pas nécessaire de prendre en compte toutes les dimensions. On en considère un nombre plus faible que celui de l'espace d'origine. Ceci présente bien sûr des difficultés car il faut déterminer les dimensions à conserver mais il faut aussi que les données se prêtent bien à la réduction des dimensions.

5.2.3.5 Contraction

La contraction d'un espace E permet de ne pas conserver d'informations à propos des cellules vides. L'index ne renseigne plus que sur E réduit aux cellules non-vides. Cette propriété est satisfaite par la plupart des méthodes d'indexation. Cependant, d'après Freeston [61], les méthodes basées sur une transformation de E en un espace linéaire ordonné (par exemple le Z ordering [107] ou le Grid File, traité en section 5.2.2.5) en sont dépourvues.

Dans un espace de dimension d, la subdivision la plus simple partitionne l'espace en deux sous-espaces sur chaque dimension. Il y a donc au minimum 2^d cellules. Ce nombre de cellules devient non-négligeable pour une certaine valeur de d et dépasse très rapidement le nombre d'objets. Il est donc important pour un index de posséder cette propriété.

5.2.3.6 Pouvoir de séparation

Cette caractéristique est liée à l'organisation physique des nœuds de l'arbre (voir section 5.2.3.7). Elle désigne la proportion d'objets définitivement écartés à chaque étape de progression dans l'arbre lors d'une recherche. Elle est liée au *fan-out* (envergure) de chaque nœud : plus le nombre de sous-espaces référencés dans un nœud est élevé, plus le pouvoir de séparation est grand.

Les travaux sur les index négligent les traitements effectués en mémoire centrale et accordent plus d'importance aux accès disque. Un bloc (ou page) disque chargé en mémoire offre un certain nombre d'entrées d'index, contenant chacune des informations sur le sous-espace représenté. La détermination du sous-espace contenant l'objet recherché étant considérée négligeable en termes de coût, il est important que la page d'index offre le maximum d'informations possible. Pour une même taille de bloc physique, une technique discriminant les nœuds fils sur une dimension (par exemple, le k-D-B-Tree [119] ou l'Hybrid-Tree [33]) a un fan-out plus important qu'une technique utilisant plus de dimensions (c'est le cas des autres techniques arborescentes). Ceci s'explique par le fait que les données décrivant des cellules sur plusieurs dimensions occupent un espace de stockage plus grand que la description du même nombre de cellules sur une dimension : sur une même page, on peut donc stocker moins de descripteurs multidimensionnels que de descripteurs monodimensionnels (voir la discussion en section 5.2.4.2 plus loin).

Le fan-out est un paramètre important de la forme de l'arbre pour un même jeu de données. S'il est faible, la profondeur de l'arbre augmente. Il en est de même pour le nombre de nœuds visités lors d'une recherche. La performance générale de la technique en est affectée, sachant

qu'une recherche intervient lors de l'interrogation, mais aussi lors des insertions et suppressions, soit lors de toute opération sur les données.

5.2.3.7 Taux d'occupation

Rappelons que les techniques arborescentes allouent à chaque nœud un espace de stockage. L'occupation de cet espace varie au cours des opérations sur l'arbre (créations, insertions et suppression d'objets). À chaque instant, l'espace alloué est partiellement ou complètement utilisé. La proportion utile de cet espace pour un nœud représente son « taux d'occupation ». En général, le taux d'occupation désigne une valeur plancher, un minimum de la proportion utile (*minimum occupancy*) d'un nœud garanti par la technique. Par exemple, cette proportion garantie est de 50% pour le B-Tree [5].

Pour les techniques multidimensionnelles, un taux d'occupation minimum semble difficile à obtenir. Néanmoins, le BV-Tree de Michael Freeston [61] garantit un taux de 33%.

5.2.3.8 Flexibilité

Un aspect important d'une technique d'indexation est, au-delà des principes de conception, l'organisation de l'index sur disque et en mémoire. Depuis le B-Tree [5], les techniques d'indexation allouent à chaque nœud d'un arbre un espace de stockage constant ou variable sur disque.

La flexibilité d'une technique s'entend par cette variabilité de la taille des nœuds. Contrairement à certains domaines d'applications, la flexibilité n'est pas recherchée ici car elle est préjudiciable à l'efficacité d'une technique. En effet, des tailles variables pour les nœuds de l'arbre, combinées avec un taux d'occupation non garanti, empêchent de prédire la taille de l'arbre, et par conséquent, de garantir ses performances.

On note que le LSD-Tree [81] décrit un arbre à plusieurs niveaux logiques organisé de telle sorte qu'un niveau réside en mémoire centrale, et les autres en mémoire secondaire.

5.2.3.9 Complexité

Les complexités temporelle et spatiale forment le critère d'évaluation le plus important d'une technique d'indexation car l'objectif d'un index est de réduire les temps de réponse lors du traitement des requêtes. En raison de la latence des supports de stockage vis-à-vis du processeur, cet objectif revient à réduire le nombre d'opérations d'entrées/sorties. D'après Frees-

ton [61], la technique d'indexation multidimensionnelle idéale devrait préserver les propriétés de l'arbre B :

- il existe un chemin d'accès unique à un objet ;
- une mise-à-jour se fait en temps logarithmique par rapport au volume de données ;
- les modifications structurelles de l'arbre sont locales et non globales ;
- un taux d'occupation (voir section 5.2.3.7) minimum est garanti.

De telles propriétés permettent de fixer des limites, en temps comme en espace de stockage, aux besoins d'une technique particulière et de la rendre « entièrement prévisible » et donc utilisable pour des applications critiques.

Les complexités théoriques des index sont très rarement disponibles dans les publications correspondantes. On notera que les techniques d'indexation multidimensionnelle ont une complexité spatiale exponentielle dans le pire des cas. Cependant, on relève quelques exceptions dans la littérature, liées à des techniques procédant par transformation de l'espace multidimensionnel vers un espace monodimensionnel traité par un arbre B+. Par conséquent, elles héritent des propriétés de l'arbre B+. C'est ainsi que le Pyramid-Tree [12] (voir section 5.2.2.2) et l'UB-Tree [4] ont une complexité spatiale linéaire en fonction du nombre d'objets à indexer, et une complexité temporelle logarithmique.

Weber, Schek et Blott montrent dans [147] que les techniques utilisées pour la recherche de plus proches voisins ont une complexité temporelle linéaire en fonction du volume de données. Ils mentionnent, comme résultat de l'expérimentation menée, que la recherche séquentielle est meilleure que toutes les méthodes d'indexation testées à partir de 10 dimensions pour la recherche de plus proches voisins.

5.2.3.10 Équilibre

Cette propriété est généralement entendue comme la faculté d'une structure à placer les feuilles de l'arbre à une même distance de la racine. Mais Michael Freeston, dans son article sur le BANG File, la définit différemment (voir section 5.2.2.5) : l'équilibre devient la propriété d'un arbre à placer dans ses branches gauche et droite à peu près le même nombre d'objets.

L'équilibre de l'arbre d'indexation, lorsqu'il désigne la hauteur des feuilles, permet de borner sa profondeur, d'estimer sa taille et les temps de réponse. Mais Berrani *et al.* [14] montrent que l'équilibre de l'arbre n'implique pas automatiquement que la complexité est calculable et vice-versa.

5.2.3.11 Réglages

L'existence de paramètres dans une technique d'indexation pose un problème difficile, celui de leur détermination. Un fonctionnement optimal (par rapport à d'autres caractéristiques comme la complexité ou l'équilibre de l'arbre) requiert souvent d'adapter les paramètres au jeu de données. À ce titre, les réglages semblent « porter préjudice aux techniques paramétrées » d'après Zobel *et al.* [155]. Le X-Tree [13] est un exemple de technique à paramètres pouvant dégénérer en d'autres structures moins performantes si le paramètre est mal fixé (voir [14] pour plus de détails).

5.2.3.12 Stabilité de la structure de données

Cette caractéristique se réfère à l'augmentation en volume de l'index en fonction de la quantité de données. Dans certaines techniques, la simple insertion d'un objet peut provoquer le doublement en taille de la structure d'index. C'est le cas pour la technique EXCELL [133]. D'autres techniques inspirées du Grid File [105] (section 5.2.2.5) souffrent également d'une croissance rapide du nombre de cellules indexées, la croissance étant cette fois sans rapport avec le nombre de dimensions.

Cette croissance du volume des structures d'index survient pour les techniques dans lesquelles les partitionnements causés par une insertion ont une portée globale plutôt que locale.

5.2.3.13 Coût CPU

La plupart des techniques semblent se préoccuper d'éléments de performance comme la profondeur des arbres, la complexité spatiale et temporelle ou le nombre d'accès disque, mais pas du coût CPU induit par les calculs inhérents aux techniques. Bien au contraire, le M-Tree [37] vise à « réduire le coût CPU des calculs de distance » par l'utilisation de l'inégalité triangulaire. La technique évite d'effectuer les calculs de distance, particulièrement coûteux dans de grandes dimensions, qui se révéleraient inutiles.

5.2.4 Discussion

5.2.4.1 Types de recherches

Les types de recherches que permet une technique d'indexation dépendent de l'organisation de la structure d'index. La recherche exacte est la plus simple à implémenter puisqu'elle consiste en une simple comparaison avec les critères de sélection. La recherche par intervalles,

qui peut être vue comme une extension de la recherche exacte à plusieurs valeurs sur un domaine discret, est moins fréquente, mais plus que la recherche par similarité ou la recherche des plus proches voisins. Ces deux derniers types de recherches sont particulièrement difficiles dans les espaces de grandes dimensions car ces recherches se fondent sur des distances ou des mesures de similarité. Or le nombre de cellules voisines pour une cellule quelconque augmente exponentiellement avec la dimension. De même, la probabilité pour un point p de l'espace d'être près d'une frontière tend vers 1 : la probabilité de devoir explorer les cellules voisines lors de la recherche des plus proches voisins de p tend vers 1 [14, 147]. Enfin, la croissance de la dimension implique la définition d'intervalles de plus en plus grands pour spécifier un même volume. Ainsi, dans le plan, un hypercube de dimension 2 (un carré donc) dont chaque côté a une longueur de 10% sur chaque dimension occupe 1% de l'espace total. Mais dans un espace à 10 dimensions, un hypercube qui occupe 1% du volume a un côté de longueur $l = \sqrt[10]{0.1}$, soit environ 63% du domaine. Ces trois facteurs (le nombre de cellules, la proximité des frontières et l'extension des domaines couverts) font que les recherches par similarité ou distance exigent le parcours d'une grande partie de l'espace indexé. D'après Weber, Schek et Blott dans [147], les techniques d'indexation implémentant ces index deviennent moins performantes que la recherche séquentielle dès 10 dimensions.

5.2.4.2 Stockage des descripteurs de cellules

Les structures d'index en arbre allouent à chaque nœud une *page* (espace de stockage), correspondant à une (ou plusieurs) unités d'allocation du système d'exploitation et du SGBD. Supposons une taille de page de 8 Ko. En faisant abstraction de toute métadonnée, les données stockées dans une page d'index sont des entrées d'index. Une entrée d'index offre le moyen de décrire de manière non ambigüe le sous-arbre, éventuellement réduit à une feuille, auquel elle correspond. Mais une entrée contient également un pointeur vers la page correspondant au sous-arbre. Dans un arbre B+, les informations sur un sous-arbre se réduisent à une valeur de l'attribut indexé, soit 4 octets pour des entiers. Le pointeur est généralement aussi un entier de la même taille, soit un total de 8 octets par entrée. Une page d'index de 8 Ko dans un arbre B+ contient donc 1024 entrées par page. Ceci autorise 2^{20} $(1.048.576)$ enregistrements pour un arbre à deux niveaux et 2^{30} $(1.073.741.824)$ enregistrements pour un arbre à 3 niveaux d'indexation.

Dans un index multidimensionnel, un intervalle sur l'une des dimensions est nécessairement représenté par deux valeurs. Pour un espace à $d = 8$ dimensions entières, la représentation complète d'une cellule requiert d points, soit 64 octets, et celle d'une entrée d'index (qui prend en compte un pointeur de page), 68 octets. Ceci permet 15 entrées par page d'index. Un arbre

plein et équilibré sur 3 niveaux autorise donc $15^3 = 3.375$ cellules, sur 4 niveaux, $15^4 = 50.625$ cellules. Ces valeurs sont en retrait par rapport à une subdivision régulière de l'espace par n hyperplans par dimension. En effet, les nombres de cellules obtenues seraient de $(n+1)^8$ cellules, soit $3^8 = 6.561$ et $4^8 = 65.536$ cellules pour une subdivision par deux et trois hyperplans respectivement. On constate ainsi que les index multidimensionnels ont une hauteur relativement plus grande que les index monodimensionnels pour un même nombre d'enregistrements indexés. Ce fait peut être atténué par des représentations incomplètes ou hiérarchisées des cellules, ou par l'utilisation de techniques de compression. Mais en règle générale, un index multidimensionnel requiert plus d'accès disque qu'un index monodimensionnel pour une recherche exacte d'un même enregistrement.

5.2.4.3 Recouvrement

On constate que les index multidimensionnels ne sont pas groupés (voir section 5.1 pour la définition d'un index groupé). Autrement dit, les enregistrements ayant des valeurs proches ne sont pas nécessairement voisins sur le support physique. En général, une unité d'indexation élémentaire (une feuille) regroupe les objets dont les valeurs sur les attributs d'indexation sont proches. Si la distinction entre nœuds se fait sur la base de d dimensions, toute sélection sur k attributs avec $k < d$ réalise une projection de l'espace de données initial sur un espace de dimension inférieure. Ce phénomène est une autre déclinaison des collisions des fonctions de hachage (voir section 5.1.4) et des projections sur la droite ou le plan (figure 5.15). Une sélection sur k dimensions désigne implicitement plusieurs unités d'indexation de même que le point $p1$ de la figure 5.15 désigne plusieurs points de la droite (a, d).

Les feuilles correspondant à une sélection sur k dimensions sont éparpillées dans l'arbre. Des critères particuliers peuvent permettre de circonscrire les résultats à une région de l'arbre d'indexation, mais l'expérience n'est pas reproductible sur tous les critères. Dans le cas général, les feuilles sélectionnées se retrouveront sur toute la largeur de l'arbre. Cependant, une sélection sur d attributs désignera une unique feuille pour les techniques sans recouvrements. Par conséquent, le chemin suivi par la recherche est unique et le nombre de nœuds parcourus, équivalent au nombre d'accès disque, est minimal. Dans les autres cas de sélections, la recherche est moins efficace : les feuilles résultats sont dispersées en cas de sélection sur un nombre de dimensions inférieur à d et, en cas de recouvrement, certains des nœuds visités apportent des informations redondantes.

Par conséquent, il est préférable qu'une technique d'indexation n'admette pas de recouvrement. On évite ainsi des opérations (recherches, insertions, modifications et suppressions) peu

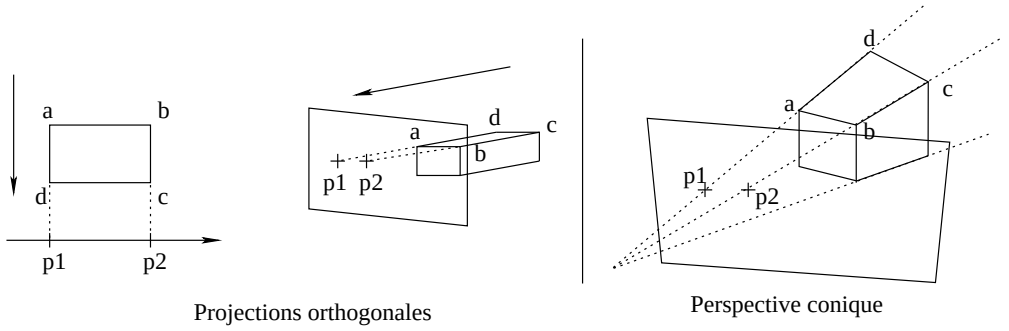


Figure 5.15 – Projections sur des espaces de dimensions inférieures

efficaces. Mais plus que l'aspect géométrique de la notion de recouvrement, c'est la possibilité de chemins multiples vers un même objet (par exemple, $b1$ dans la figure 5.9 en section 5.2.2.1), synonyme d'entrées multiples, qui est à éviter.

5.2.4.4 Performance

À propos de la performance, on note que les méthodes par hachage devancent les méthodes en arbre grâce au nombre d'accès disque constant (voir sections 5.1.4 et 5.2.2.5). Malheureusement, les méthodes d'indexation par hachage ne sont efficaces que pour les recherches exactes et les fonctions adéquates sont difficiles à trouver dans un contexte dynamique. Aucune technique (arborescente ou par hachage) ne répond à tous les besoins qui motivent l'utilisation d'index multidimensionnels. Même le Pyramid-Tree, qui est la seule technique échappant à la malédiction de la dimensionnalité (voir section 5.2.2.2), ne traite efficacement que les données uniformes et les requêtes par intervalles. Berchtold *et al.* montrent dans [11] qu'il n'existe pas, pour des données uniformes, une structure d'index qui soit performante sur toutes les dimensions pour les recherches de plus proches voisins. Dans [147], les auteurs vont plus loin et considèrent qu'il vaut mieux tenter d'améliorer la recherche séquentielle que de « se battre pour une guerre déjà perdue » contre les difficultés des grandes dimensions (dont on peut trouver une description dans le même article). Bien que ces conclusions ne concernent que la recherche de plus proches voisins, l'examen des propositions de techniques d'indexation dans leur ensemble (16 techniques recensées entre 1990 et 1999 contre 3 [10, 113, 144] entre 2000 et 2007) semble les confirmer.

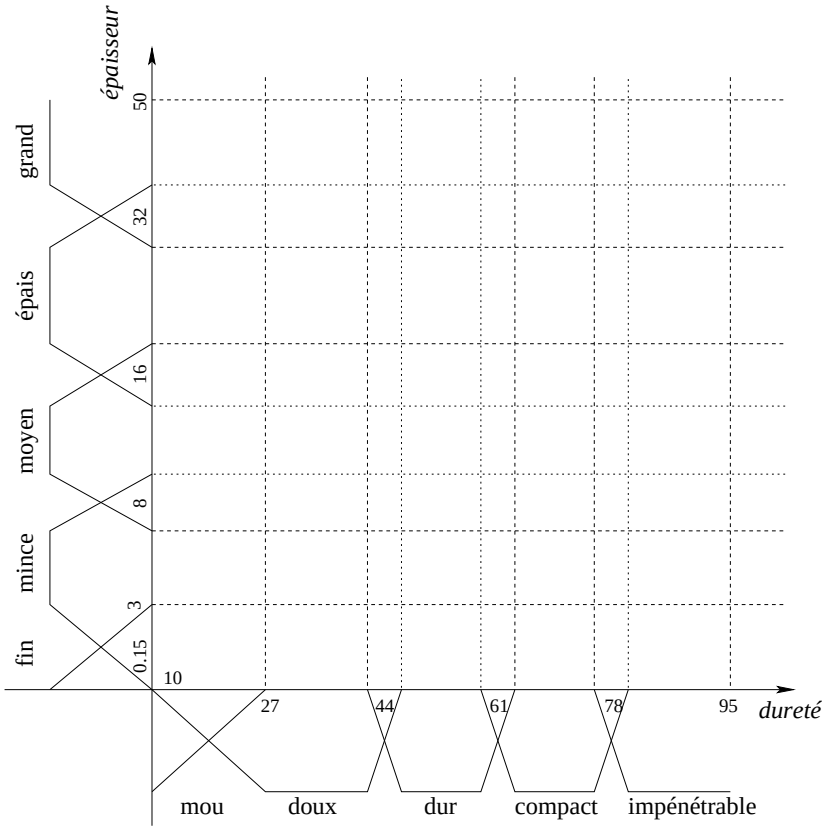


Figure 5.16 – Partitionnement de l'espace dans SAINTETIQ

5.2.4.5 Résumés SAINTETIQ

Au vu des techniques d'indexation multidimensionnelles disponibles, les résumés se rapprochent des techniques issues du k-d-Tree. Outre l'organisation hiérarchique et l'ordre partiel qui en découle, typiques des index arborescents, la subdivision de l'espace est statique et se fait par plans parallèles aux axes. En conséquence, les cellules sont des hyperrectangles. La construction de l'arbre par insertions successives est également indéterministe (au sens de la section 5.2.3.1).

On retrouve néanmoins quelques différences au sein des résumés. En particulier, les cellules sont de taille fixe (figure 5.16), il n'y a donc pas de volumes englobants mis-à-jour en fonction des données qui y sont référencées. Les nœuds d'une hiérarchie de résumés sont des nœuds logiques. Leur contenu n'est pas fixé par une contrainte telle que la taille d'une page physique. Les sujets liés à cette taille, relevant de la politique de partitionnement, ne se posent

pas ici. Comme technique d'indexation, les résumés seraient une technique par point. Enfin, une hiérarchie de résumés est un arbre non équilibré.

Table 5.1 – SAINTETIQ par rapport aux propriétés des techniques d'indexation

Propriété	Impact	SAINTETIQ	Bilan
Indéterminisme	-	$\surd$	-
Forme des régions		Hypercubes	
Recouvrement	-	$\surd$	-
Réduction de la dimension	+	($\surd$)	(+)
Contraction	+	$\surd$	+
Pouvoir de séparation		N/A	N/A
Équilibre	+	$\times$	-
Réglages	-	$\times$	+
Taux d'occupation garanti	+	N/A	N/A
Flexibilité	-	$\surd$	-
Stabilité		< nombre de nœuds	
Coût CPU	+	$\times$	-

Notons enfin qu'il suffit d'une relation d'ordre partiel comme base pour un index en arbre. C'est ainsi que Hellerstein *et al.* [79] ont proposé un index général capable d'implémenter tout index arborescent équilibré. De manière simplifiée, définir une technique d'indexation dans ce cadre revient alors à définir la sémantique de l'ordre partiel et la politique de partitionnement. Il est ainsi possible de simuler le comportement d'un index B+ ou R-Tree.

Dans le reste de cette section, les résumés sont analysés vis-à-vis des propriétés de la section 5.2.3, dans l'hypothèse qu'une hiérarchie de résumés constitue une structure d'index. On peut ainsi déterminer, d'une part, les acquis des résumés SAINTETIQ, et d'autre part, les caractéristiques auxquelles porter une attention particulière afin d'atténuer ou d'annuler leur impact négatif. Le tableau 5.1 propose un récapitulatif des caractéristiques recensées dans la littérature. Ce tableau indique également, le cas échéant :

- l'impact positif ou négatif (matérialisé respectivement par « + » et « - ») de la propriété considérée sur les performances des techniques ;
- si la propriété s'applique à SAINTETIQ (« $\surd$ » si oui, « $\times$ » si non) ;

- le bilan de cette propriété pour les résumés, négatif (« - »), positif (« + ») ou positif sous conditions (« (+) »).

Taux d'occupation garanti. Cette caractéristique n'est pas applicable aux résumés car les nœuds d'une hiérarchie sont des nœuds logiques, sans espace alloué sur le support de stockage. La question de la capacité effective ne se pose donc pas encore.

Pouvoir de séparation. De même que le taux d'occupation, le pouvoir de séparation n'est pas encore applicable. Mais quelques observations peuvent être faites. D'abord, il n'existe pas dans le processus de construction des résumés de politique relative au nombre de dimensions suivant lesquelles distinguer les niveaux inférieurs. Il est effectivement possible de retrouver, au sein d'une même hiérarchie, des nœuds dont les fils diffèrent sur des attributs différents.

Indéterminisme. La question de la sensibilité à l'ordre de SAINTETIQ a fait l'objet d'une discussion par R. Saint-Paul dans [125]. L'auteur conclut que « la topologie de la hiérarchie n'est pas primordiale selon les applications envisagées ». Il ressort de son analyse que l'effet d'ordre est inévitable en raison, notamment, du caractère incrémental de la construction. De plus, les opérateurs d'apprentissage, bien que capables de défaire et refaire une topologie, ont une action dont la portée est très limitée au sein de la hiérarchie. Dans un index, la topologie de la structure est synonyme des liens père-fils entre nœuds physiques. L'impact de l'indéterminisme pourrait être négatif sur l'espace total occupé par l'index et par conséquent sur les temps de réponse, en induisant par exemple un nombre moyen d'accès disque plus important.

Recouvrement. Le recouvrement des unités d'indexation est une caractéristique qu'on tente habituellement d'éviter dans un index. Cependant, la notion de gradualité inhérente aux descriptions par variables linguistiques requiert qu'une valeur du domaine puisse appartenir simultanément à deux descriptions distinctes. Dans un contexte d'utilisation de descriptions linguistiques, le recouvrement est nécessaire et ne doit donc pas être perçu comme un désavantage.

Réduction de la dimension. Cette propriété est en réalité extrinsèque aux hiérarchies de résumés. Il est en effet possible de projeter une relation $\mathsf{R} = (A_1, \ldots, A_n)$ sur un nombre d'attributs (m) inférieur à la dimension (n) de la relation. On obtient alors une relation $\mathsf{R}^* = (A_1, \ldots, A_m)$ sur laquelle les résumés seront construits.

Flexibilité. Comme le taux d'occupation, la flexibilité, qui désigne le caractère variable des tailles d'entrées d'index, s'applique aux nœuds physiques. On peut néanmoins déjà noter que la représentation en intension d'un résumé a une taille variable : les résumés feuilles présentent un unique couple degré-étiquette par attribut alors que les résumés de niveaux supérieurs deviennent multivalués. L'impact négatif de la flexibilité peut être annulé en adoptant une autre représentation de l'intension. Par exemple, l'assignation d'une position binaire invariable à chaque descripteur d'une variable linguistique (comme effectué en section 4.3.3) permet de représenter une intension sur un nombre fixe de bits. On peut ainsi arriver à homogénéiser la taille des résumés à tous les niveaux d'une structure d'index.

Équilibre. La propriété d'équilibre est importante dans les systèmes contraints en temps de réponse. Un temps de traitement court est un atout pour une technique d'indexation, mais l'interrogation flexible n'est habituellement pas un processus temps-réel. La contrainte d'équilibre peut donc être relaxée.

Stabilité. Ici encore, la stabilité se réfère à un aspect matériel du stockage de la structure d'index, appelant des remarques. Saint-Paul établit, analytiquement et expérimentalement dans [125], que les modifications structurelles d'une hiérarchie de résumés interviennent majoritairement au début de sa construction. De plus, Raschia montre dans [115] que la taille d'une hiérarchie de résumés est bornée, indépendamment du nombre d'enregistrements à résumer. Outre ces considérations, on note que les résumés ne représente pas l'espace vide. Il n'y a donc pas de risque que le nombre d'unités d'indexation augmente comme cela peut être le cas dans la technique du Grid File (voir section 5.2.2.5) où la subdivision de l'espace est également indépendante des données.

Coût CPU. Le coût CPU doit être pris en compte lorsque les calculs nécessaires pour mener les recherches et autres opérations de l'index ne sont plus négligeables face aux temps d'accès disque. C'est le cas dans les très grandes dimensions. Il n'y a pas de calculs de ce type lors d'un parcours de hiérarchie SAINTETIQ. A priori, cette caractéristique ne sera pas pénalisante, mais il reste à le confirmer expérimentalement.

5.3 Conclusion

Ce chapitre a présenté un état de l'art des techniques d'indexation. Un index désigne toute structure de données conçue pour accélérer les traitements effectués sur un ensemble de données. Pour sa construction, des informations issues des données sont nécessaires. On parle d'index monodimensionnel lorsque ces informations portent toutes sur un même domaine d'attribut, et d'index multidimensionnel lorsque ces informations proviennent de plusieurs attributs.

Les index monodimensionnels présentés sont soit à base de répertoires (c'est le cas des tables de hachage), soit organisés en arbre (ISAM et arbres B). La recherche sur les index à une dimension semble avoir abouti à la conclusion que l'arbre B+ est le meilleur index possible.

Les index multidimensionnels sont également à organisation linéaire pour les techniques qui effectuent un hachage indirect sur plusieurs dimensions, ou à organisation arborescente pour toutes les autres, qui fixent un ordre partiel sur les unités d'indexation. Dans ce chapitre, les structures de données à la base des index arborescents ont été présentées (K-D-Tree et BSP-Tree), de même que les techniques marquantes dont les propriétés ou les performances sont singulières (par exemple, le Pyramid-Tree, dont le nombre de cellules évolue de façon linéaire plutôt qu'exponentielle avec la dimension). Ensuite, les propriétés mathématiques et structurelles des index multidimensionnels, recensées dans la littérature, ont été détaillées. Ces propriétés couvrent les aspects conception (contraction, réduction de dimension, complexité, ...) et implémentation (flexibilité, réglages, taux d'occupation, ...) des techniques. Les conclusions qui découlent de cette présentation des techniques d'indexation ont été finalement exposées. Elles permettent de mettre en évidence des similarités entre ces index et les résumés du modèle SAINTETIQ, justifiant ainsi le travail d'implémentation du chapitre suivant.

CHAPITRE 6

Implémentation des résumés SAINTETIQ en tant que méthode d'accès

Introduction

Ce chapitre applicatif décrit l'implémentation des résumés SAINTETIQ en tant que méthode d'accès au sein du système de gestion de bases de données PostgreSQL. L'interrogation des résumés envisagée jusqu'ici consistait à décrire des enregistrements, sans accéder à la relation, et donc, sans utiliser les valeurs d'attributs. Cependant, le chapitre 3 montre que les systèmes d'interrogation flexible ont parfois besoin des valeurs d'attributs, en particulier pour déterminer l'adéquation de chaque enregistrement résultat à la requête traitée. A-t-on la possibilité d'utiliser une hiérarchie de résumés pour extraire des valeurs d'attributs ? On suppose que oui car les résumés conservent les identifiants des enregistrements décrits. Nous étudions ici la faisabilité et l'efficacité en temps de réponse de cette possibilité, sous le SGBD PostgreSQL.

Les raisons qui ont motivé le choix de PostgreSQL sont les suivantes. D'abord, PostgreSQL est très utilisé dans le monde de la recherche en bases de données et y a bonne presse. De plus, en tant que logiciel libre, le code source est aisément disponible. Ce point en particulier a très largement facilité la mise en œuvre de notre implémentation, grâce à la haute qualité du code publié, et aux exemples qu'ont été les méthodes d'accès standard du système. Ensuite, le SGBD est explicitement orienté « plug-in » afin de présenter une grande versatilité sans pour autant céder du terrain sur le plan de l'efficacité. Il admet l'intégration de nouvelles fonctionnalités par le biais de modules externes supplémentaires, éventuellement conçus par les utilisateurs. En particulier, de nouvelles méthodes d'accès peuvent être rajoutées au SGBD et bénéficier

d'un couplage fort avec le noyau du système de la même manière que les méthodes d'accès standard. Ceci garantit le maximum de performance possible, élément important pour les index, dont le rôle est d'accélérer les recherches. Enfin, la documentation disponible est relativement exhaustive et une communauté de personnes, centrée sur le logiciel, est active à travers des listes de diffusion et des forums de discussion. Ce dernier aspect collaboratif est également important lors de l'ajout de plug-in sur une plate-forme logicielle spécifique.

En raison de sa fréquence, la recherche est l'opération la plus importante d'un index : elle intervient aussi bien lors des interrogations que lors des ajouts, suppressions et modifications. C'est à ce titre que l'opération de recherche est le point central de la présente étude. L'objectif n'est pas de proposer une nouvelle technique d'indexation générale dont les performances seraient meilleures que celles des techniques existantes (voir la section 5.2.4.5 et plus particulièrement, 5.2.4.5). Cependant, de par l'aspect sémantique affirmé des résumés SAINTETIQ, nous comptons montrer qu'ils sont aptes à aider les processus plongés dans le même cadre sémantique, notamment les outils d'interrogation flexible. Les conclusions tirées de cette étude permettront, à terme, de modifier la construction des résumés, et de l'adapter en vue d'une optimisation pour le rôle d'index. À ce titre, les regroupements effectués par le processus actuel sont conservés : la hiérarchie de résumés est réutilisée telle quelle. La seule modification effectuée est un changement de format, du texte simple d'un fichier XML vers une version binaire dont la section 6.2 donne le détail.

6.1 Description de l'implémentation

6.1.1 Modification de l'algorithme de recherche

Rappelons que l'interrogation précédemment décrite admet en entrée des critères exprimés sur les étiquettes linguistiques. En sortie, le processus fournit des résumés, issus de différents niveaux de la hiérarchie de résumés, ou pris uniquement parmi les feuilles. Le processus d'interrogation décrit dans le chapitre 2 peut être repris tel quel dans une méthode d'accès. Cependant, deux étapes supplémentaires s'avèrent nécessaires en entrée et en sortie du processus (voir figure 6.1) pour obtenir des enregistrements.

Dans le rôle d'index, où le service rendu est l'identification d'enregistrements, l'interrogation est intégrée à un système de gestion de bases de données (SGBD). Les hypothèses et pratiques en usage dans les SGBD doivent être respectées, notamment en ce qui concerne les requêtes. En effet, le langage de requête étant SQL, les conditions de sélection de la clause

WHERE (par exemple, `température` $= 800$) doivent être traduites en étiquettes linguistiques. Ainsi, le critère précédent devient `température` IN (*modéré*).

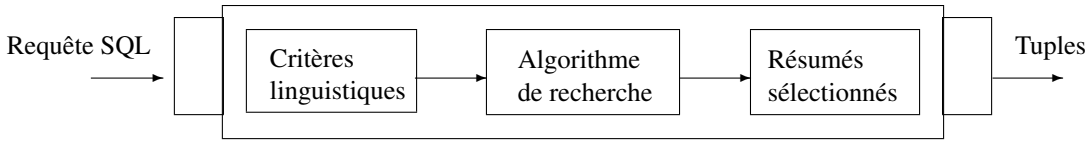


Figure 6.1 – Schéma de l'interrogation à but d'identification

En sortie, le résultat habituel d'une interrogation dans un SGBD est une liste d'enregistrements. L'extension des résumés est mise à profit pour remplacer chaque résumé (résultat de l'interrogation de résumés) par les enregistrements couverts. Mais ce faisant, d'éventuels enregistrements incorrects sont inclus. Un descripteur couvre en effet plusieurs valeurs du domaine ; il n'est pas garanti que deux tuples t_1 et t_2 d'un même résumé partagent les mêmes valeurs sur les domaines d'attributs initiaux. Ainsi, t_1 tel que $t_1.\texttt{température} = 810$ ne devrait pas faire partie du résultat de la requête SQL de critère `température` $= 800$. Pour assurer l'adéquation de l'interrogation et fournir les mêmes résultats que toute autre méthode d'accès, une étape de filtrage par rapport aux critères SQL est donc rajoutée. Son rôle est de faire la part entre les résultats valides vis-à-vis des critères SQL et les autres enregistrements incorporés du fait de leur description par les critères linguistiques, plus généraux.

Pour une interrogation à but d'identification de tuples, l'algorithme 1 (38) est modifié de manière à ne renvoyer que des feuilles de l'arbre comme résultat (voir l'algorithme 4). Il suffit d'ajouter toutes les feuilles d'un nœud interne s'il est déjà un résultat (cas 2 en section 2.2.2). Si le nœud examiné est une feuille, il est rajouté à la liste des résultats.

6.1.2 Considérations spécifiques à PostgreSQL

En raison de sa conception visant à favoriser la modularité, PostgreSQL [135] est un système « *catalog-driven* » : tous les objets reconnus par le SGBD sont référencés selon leur nature dans des tables relationnelles. C'est ainsi que les types, les opérateurs, les fonctions (qu'elles soient des procédures stockées SQL ou des routines compilées en format binaire), les classes d'opérateurs, les méthodes d'accès, les statistiques, etc. sont définis (ou pour le moins représentés) par des enregistrements. Les tables concernées ont toutes un nom de la forme « `pg_xxx` », préfixé par « `pg` », où `xxx` désigne la nature des objets : par exemple, `pg_proc` pour les fonctions (*procedures*) et `pg_am` pour les méthodes d'accès (*access methods*).

Algorithme 4 Fonction RechercheFeuille(z, $\mathcal{C}$)

$L_{res} \leftarrow$ < >
si $Corr(z, \mathcal{C}) =$ *excès* **alors**
  pour chaque nœud fils z_{fils} de z **faire**
    $L_{res} \leftarrow L_{res} + \text{RechercheFeuille}(z_{fils}, \mathcal{C})$
  fin pour
sinon
  si $Corr(z, \mathcal{C}) =$ *exacte* **alors**
    si z est une feuille **alors**
      ajouter(z, L_{res})
    sinon
      $\text{AjouterFeuilles}(z, L_{res})$
    fin si
  fin si
fin si
retourner L_{res}

Dans la terminologie PostgreSQL, une méthode d'accès est un ensemble *algorithmes & structures de données* permettant d'accéder directement à un enregistrement qui répondrait aux critères d'une requête. Dans le scénario de traitement « normal », la requête est émise par un logiciel client et traitée par le logiciel serveur, en exécution dans le même environnement logiciel ou dans des environnements distants. Elle précise les critères de recherche permettant d'identifier les objets (tuples) satisfaisant tous les critères. Comme dans tout langage, la requête subit une analyse syntaxique à l'issue de laquelle le nom de la relation concernée, les champs à afficher et les critères de sélection sont extraits. Le « moteur » du SGBD, connaissant ainsi la relation spécifiée dans la requête, en inspecte les métadonnées. Lorsqu'il y trouve un index sur les champs des critères de sélection, il abandonne le comportement par défaut (c'est-à-dire la recherche séquentielle) pour la méthode d'accès associée à l'index. Ce choix est fait parce que la méthode d'accès est supposée plus performante. Cependant, cette supposition peut être démentie par les statistiques d'usage que tient le système. Si tel est le cas, la méthode d'accès pourrait ne plus être invoquée pour ces champs.

Le rôle de la méthode d'accès est de rechercher les tuples adéquats vis-à-vis des critères de sélection. La seule exigence du SGBD à cet égard est qu'elle fournisse les informations nécessaires pour localiser un tuple au sein de sa relation. Pour PostgreSQL, ces informations

sont désignées indifféremment par les termes « *tuple ID* » et « *TID* ». Un TID est une adresse physique sur disque composée de deux éléments, le *block number* et l'*item number*.

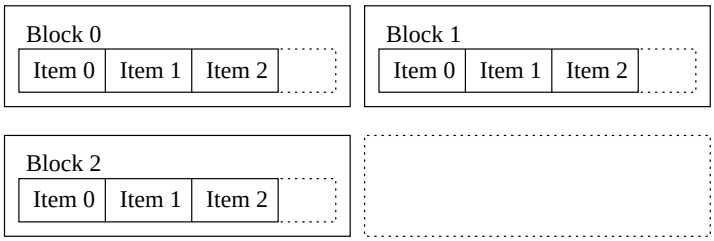


Figure 6.2 – Structure logique des fichiers de données PostgreSQL

Le système de stockage du SGBD « découpe » les fichiers disque en sections ou pages dont la taille prédéfinie (en général 8 Ko) est indépendante de celle du système de fichiers du système d'exploitation. Un *block number* (BN) est un numéro séquentiel se référant à l'une de ces sections (voir figure 6.2). Un *item number* (IN) est un numéro séquentiel au sein d'une section ; ce numéro varie d'une relation à l'autre en fonction de la taille des enregistrements. Ainsi, un *block* de 8 Ko dans une relation dont les enregistrements ont une longueur de 128 octets pourrait contenir jusqu'à 64 enregistrements en omettant toutes les métadonnées de la relation et celles propres à chaque page. Le premier enregistrement de cette relation exemple serait identifié par le couple $(BN = 0; IN = 0)$, le deuxième par $(BN = 0; IN = 1)$, et le 65$^{\text{è}}$ par $(BN = 1; IN = 0)$.

L'ajout d'une méthode d'indexation au noyau de PostgreSQL se fait par un plug-in. L'interface permettant l'inclusion d'index non standard, nécessaire pour ce module supplémentaire, est définie dans la section 48 de la documentation du SGBD [136], et abordée en section 6.3 de ce document.

Définir une nouvelle méthode d'accès revient à suivre quelques étapes. La première est d'écrire les fonctions requises par l'interface dans un langage de programmation respectant les conventions d'appel et de transmission de paramètres du langage C. Les fonctions compilées doivent être disponibles dans un (ou plusieurs) fichier binaire à chargement dynamique (des DLL – *Dynamic Link Libraries* – sous Windows, des SO – *Shared Objects* – sous Linux). La deuxième étape consiste à informer le SGBD de l'existence des fonctions (voir section 6.3 ci-après), qui entrent alors dans le catalogue des fonctions connues. La méthode d'accès n'est définie qu'après la troisième étape qui spécifie les fonctionnalités offertes et qui regroupe toutes les fonctions en un ensemble logique cohérent lié à la méthode d'accès.

L'invocation d'une méthode d'accès par le SGBD intervient lorsqu'un index est créé ou lorsqu'une mise-à-jour est faite sur une table existante. Suivant l'opération, l'appel adéquat est fait à la fonction de création de l'index, de suppression d'un tuple ou d'insertion d'un tuple. Les fonctions de la méthode d'accès sont également sollicitées lorsqu'une requête porte sur les champs pris en charge par la méthode.

6.2 Format du fichier d'index

Cette section présente le format de stockage sur disque des résumés utilisés pour répondre aux requêtes SQL. Notons que PostgreSQL n'impose pas aux méthodes d'accès d'accéder aux structures d'index par le biais de son gestionnaire de pages ou de buffer. Il n'est donc pas fait obligation d'adopter le format de ses pages. Pour l'heure, le fichier d'index que nous utiliserons est géré, comme n'importe quel autre fichier, par le système d'exploitation. Nous faisons ainsi l'économie de la documentation sur le format des pages, les conventions d'appel du gestionnaire de pages, et les verrous dûs aux contrôles de concurrence. Ceci suffit pour le prototype expérimental décrit dans ce chapitre. Mais dans le cadre d'un module externe (*plug-in*) opérationnel, le format standard sera probablement adopté pour réaliser des comparaisons équitables avec d'autres méthodes d'accès.

Comme indiqué en section 1.4.1, chaque nœud de la hiérarchie de résumés est entièrement défini par son extension, c'est-à-dire l'ensemble des enregistrements couverts par le résumé. Dans cette mise en œuvre, l'extension n'est explicitée qu'au niveau des feuilles. Ainsi, un tuple candidat n'apparaît qu'une fois dans la représentation de la hiérarchie même s'il appartient, dans l'absolu, à l'extension de chaque nœud de son ascendance. Pour tout autre nœud interne z, l'extension est implicite et peut être facilement reconstituée : c'est l'union des extensions de toutes les feuilles dans le sous-arbre de racine z. Or, l'algorithme 4 ne sélectionne que des résumés feuilles ; conserver des extensions plus haut n'a pas d'intérêt dans ce contexte. De plus, le fait de reporter les extensions au niveau des feuilles permet de supprimer les redondances. Le volume du fichier d'index est allégé et les accès disque apportent plus d'informations utiles. Cette stratégie de report des références dans les feuilles est courante pour les index en arbre.

Le fichier d'index est un fichier binaire constitué de deux parties. La première partie contient des métadonnées (nombre d'attributs de la relation résumée et connaissances de domaine). Afin d'optimiser les opérations de comparaison des tests d'appariement (section 2.2.2), les attributs sont indicés, de même que les descripteurs des variables linguistiques. Ainsi, pour $\mathsf{R} =$

(`épaisseur`, `dureté`, `température`), les indices sont *épaisseur* $= 0$, *dureté* $= 1$, *température* $= 2$. Ceci permet de représenter les informations textuelles une seule fois.

La deuxième partie du fichier d'index stocke l'arbre des résumés dans le même ordre qu'un fichier XML : un ordre préfixé, en profondeur d'abord. La structure de données qui représente un résumé contient l'adresse du premier fils et celle du frère suivant (s'ils existent) de manière à reproduire la structure en arbre (voir figure 6.3). Elle contient également son intension, utilisée pour les tests d'appariement et, si le résumé est une feuille, son extension. On notera que le fichier d'index est une version binaire du fichier XML produit par le prototype SAINTETIQ [115, 125]. À la constitution du fichier d'index, l'adresse physique des enregistrements dans la relation n'est pas encore connue. Elle est fixée ultérieurement (voir la section 6.3). Ce sont ces mêmes adresses qui seront fournies au SGBD comme résultats de recherches.

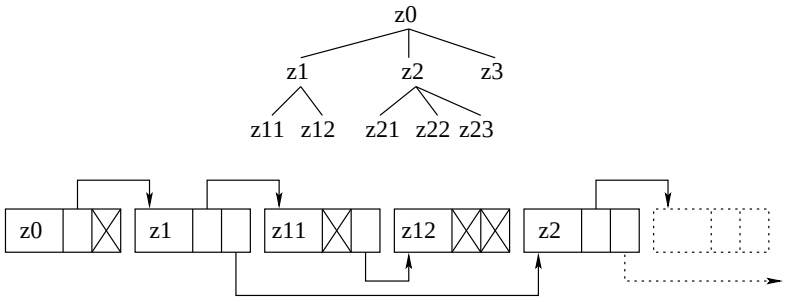


Figure 6.3 – Chaînage des entrées dans le fichier d'index

6.3 Interfaçage avec PostgreSQL

L'interface du SGBD PostgreSQL impose la définition de 12 fonctions pour une méthode d'accès [136, section 48]. Mais certaines de ces fonctions n'ont pas d'utilité pour notre expérimentation (limitée à l'interrogation). Elles ont donc été déclarées (voir tableau 6.2) mais sont sans effet. Les autres fonctions, que nous avons implémentées, sont :

- *build* : lors de la création d'un index sur une relation (par la commande CREATE INDEX), la fonction reçoit pour chaque enregistrement les valeurs de tous les champs et l'adresse physique de l'enregistrement (*TID*). Les emplacements laissés vides dans le fichier binaire (voir section 6.2) sont renseignés à cette étape ;
- *beginscan* : cette fonction reçoit les critères des requêtes SQL que l'index est en mesure de gérer. Ensuite, le noyau PostgreSQL fait un appel à la fonction suivante avant de délivrer les résultats ; il est donc préférable de déléguer le parcours de l'index à la fonction *rescan* ;

- *rescan* : cette fonction intervient si les conditions de sélection changent avant que les résultats ne soient fournis (c'est le cas de requêtes imbriquées dont un champ de la requête interne est un critère de la requête externe). Elle intervient également suite à un appel à *beginscan* ;
- *endscan* : la libération des ressources utilisées pendant l'évaluation d'une requête se fait ici ;
- *gettuple* : par cette fonction, la méthode d'accès fournit au SGBD le TID d'un enregistrement répondant à la requête évaluée ;
- *getmulti* : similaire à *gettuple* mais plusieurs TID sont fournis en une fois.

Les fonctions non implémentées ont pour rôle de :

- *markpos* : sauvegarder la position courante de parcours de l'index. Ceci est utile lorsque le parcours s'effectue en plusieurs fois et peut être interrompu ;
- *restrpos* : retourner à la position précédemment sauvegardée ;
- *insert* : référencer dans l'index un nouveau tuple, rajouté à la table indexée ;
- *bulkdelete* : supprimer les entrées d'index dont les enregistrements ont été supprimés ;
- *vacuumcleanup* : libérer les pages d'index devenues non utilisées après des suppressions ;
- *costestimate* : estimer le coût d'utilisation de l'index pour une requête dont les critères de sélection sont transmis à la fonction. Cette fonction permet au SGBD de choisir l'index le plus adapté lorsque plusieurs index existent pour la même table sur les attributs mis en jeu dans la requête.

Les fonctions sont déclarées par le biais d'instructions SQL. Pour les raisons de modularité et d'évolution déjà évoquées, l'API PostgreSQL autorise un type de données générique (« *internal* ») pour représenter tous les autres types. L'en-tête d'une fonction contient ce type pour chaque argument. Le nombre d'arguments est donc la seule information pertinente dans une signature de fonction. Par conséquent, les déclarations de fonctions dans les tableaux 6.1 et 6.2 sont abrégées et font apparaître le nombre d'arguments de chaque fonction. Cependant, l'instruction SQL soumise au SGBD doit être complète (par exemple, la fonction *build*) dans le tableau 6.1.

La méthode d'accès n'est réellement intégrée au système qu'après l'insertion d'un enregistrement adéquat dans la table `pg_am` qui fait partie du catalogue système et répertorie les méthodes d'accès disponibles. Le tableau 6.3 montre l'instruction par laquelle cette insertion est réalisée.

Table 6.1 – Déclaration des fonctions de l'interface des méthodes d'accès

Fonction	Déclaration SQL

build/3	`CREATE FUNCTION seqbuild(internal, internal, internal)`
	`RETURNS VOID`
	`AS '$libdir/seqsum' LANGUAGE C VOLATILE;`

beginscan/3	`CREATE FUNCTION seqbeginscan(...)`
	`RETURNS bool AS`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

gettuple/2	`CREATE FUNCTION seqgettuple(...)`
	`RETURNS bool`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

getmulti/4	`CREATE FUNCTION seqgetmulti(...)`
	`RETURNS bool`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

rescan/2	`CREATE FUNCTION seqrescan(...)`
	`RETURNS void`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

endscan/1	`CREATE FUNCTION seqendscan(...)`
	`RETURNS void`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

6.4 Mise en œuvre de l'index

La gestion des transactions et la gestion de la concurrence par PostgreSQL posent la contrainte suivante aux méthodes d'accès : la structure d'index ne peut être stockée en mémoire pour être réutilisée plus tard. Ceci impose un accès au fichier d'index pour chaque requête. Toutes les informations nécessaires pour répondre à une requête doivent donc y être stockées.

Les méthodes d'accès standard (tables de hachage, arbre B+, et R-Tree) dans PostgreSQL sont implémentées de telle sorte que le parcours d'une structure d'index en arbre est fait en plusieurs fois au fil des appels aux fonctions *getmulti* et *gettuple*. Il est donc nécessaire de gérer une pile des pages parcourues de manière à reprendre la recherche à l'endroit où le dernier TID d'un enregistrement résultat a été trouvé. Par souci de simplicité, nous avons choisi d'effectuer la recherche en une fois dans *rescan* ; une liste de TID est conservée en mémoire grâce à un

Table 6.2 – Déclaration des fonctions non implémentées

Fonction	Déclaration SQL

markpos/1	`CREATE FUNCTION seqmarkpos(...)`
	`RETURNS void`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

restrpos/1	`CREATE FUNCTION seqrestrpos(...)`
	`RETURNS void`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

insert/6	`CREATE FUNCTION seqinsert(...)`
	`RETURNS bool`
	`AS '$libdir/seqsum' LANGUAGE C VOLATILE;`

bulkdelete/3	`CREATE FUNCTION seqbulkdelete(...)`
	`RETURNS internal`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

vacuumcleanup/3	`CREATE FUNCTION seqvacuumcleanup(...)`
	`RETURNS internal`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

costestimate/7	`CREATE FUNCTION seqcostestimate(...)`
	`RETURNS void`
	`AS '$libdir/seqsum' LANGUAGE 'c' VOLATILE;`

Table 6.3 – Insertion de la méthode d'accès dans la table système adéquate

```
INSERT INTO pg_am
VALUES('saintetiq', 1, 0, 0, false, true, false, false, false,
'seqinsert', 'seqbeginscan', 'seqgettuple', 'seqgetmulti',
'seqrescan', 'seqendscan', 'seqmarkpos', 'seqrestrpos', 'seqbuild',
'seqbulkdelete', 'seqvacuumcleanup', 'seqcostestimate';)
```

mécanisme détaillé dans le paragraphe suivant. *getmulti* et *gettuple* se contentent par la suite d'extraire les adresses physiques de la liste. Cette dernière fait partie des ressources libérées après que tous les résultats ont été fournis.

La réutilisation d'une zone mémoire précédemment allouée pendant la session de travail est interdite en général et probablement impossible à réaliser. Mais elle est autorisée au cours de l'exécution d'une même requête. Lorsqu'une requête est reçue, la fonction *beginscan* est invoquée avec en paramètre une structure de données (*ScanDesc*, pour « *scan descriptor* ») qui décrit les critères de recherche. L'un des champs de cette structure est un pointeur (« *opaque* ») dont l'usage est réservé à la méthode d'accès. Celle-ci peut donc y référencer un espace mémoire ou une autre structure de données réutilisable par les fonctions de la méthode d'accès. Cependant, cet espace devant être libéré par la fonction *endscan*, la méthode ne peut considérer sa conservation comme un acquis : dans tous les cas, le serveur PostgreSQL remet la mémoire dans l'état précédant l'exécution de la requête.

Les classes d'opérateurs Durant le travail d'implémentation, la documentation du SGBD et le code source ont apporté réponse à toutes les questions rencontrées. L'exception notable des classes d'opérateurs (*operator classes*) leur vaut une section dans ce document. La notion de « classe d'opérateur » a en effet été très difficile à appréhender. Nous offrons ici un complément d'explication afin d'en faciliter la compréhension.

Une requête simple dans le langage SQL est sous la forme SELECT [liste_de_champs] FROM [liste_de_tables] WHERE [clause_de_sélection]. Dans le cas le plus fréquent, la clause de sélection est une condition d'égalité, par exemple dans « SELECT * FROM Employés WHERE Département = 'RH' », ou une inégalité quelconque, par exemple dans « SELECT * FROM Employés WHERE Age ≤ 20 ». La forme générale d'un critère est `champ op valeur`.

Une classe d'opérateur désigne simplement les instanciations de l'opérateur `op` que peut traiter une méthode d'accès. Cette définition se fait pour un type de données spécifique. Le tableau 6.4 donne un exemple de deux classes d'opérateurs SAINTETIQ (*« ... USING saintetiq... »*) pour les types entier et caractère. Par ces définitions, le SGBD est informé que la méthode d'accès traite les requêtes où l'opérateur reliant le champ et la valeur est l'opérateur d'égalité. Si la méthode avait admis les recherches par inégalité utilisant l'opérateur « >= » sur les entiers, la déclaration de la classe d'opérateur sur le type *int4* aurait comporté en plus `USING saintetiq AS OPERATOR 2 >=`.

Table 6.4 – Déclaration des classes d'opérateurs supportées

```
CREATE OPERATOR CLASS seq_int4_ops DEFAULT FOR TYPE int4
USING saintetiq AS OPERATOR 1 =;
```

```
CREATE OPERATOR CLASS seq_char_ops DEFAULT FOR TYPE char
USING saintetiq AS OPERATOR 1 =;
```

6.5 Expérimentation

6.5.1 Données d'expérimentation

L'expérimentation que nous détaillons ici a été menée sur plusieurs jeux de données présentés dans le tableau 6.5, correspondant chacun à une table relationnelle. La table *CIO* contient des données bancaires réelles détaillées dans [126]. Les autres tables contiennent des données générées aléatoirement sur le domaine entier $[0, 127]$ sans autre contrainte statistique qu'une moyenne centrée sur le domaine. Le choix arbitraire de ce domaine se justifie par le fait que les valeurs des champs n'interviennent pas dans le processus d'interrogation qui ne manipule que les étiquettes linguistiques. Autrement dit, ce domaine est équivalent à n'importe quel autre plage de valeurs dès lors que les variables linguistiques définies sur les domaines ont la même forme géométrique à l'échelle près.

Table 6.5 – Jeux de données

	Jeu de données			Hiérarchie de résumés		
Nom	Dimension	Termes	Tuples	Nœuds	Feuilles	Profondeur
CIO	10	34	33.733	27.304	14.269	23
d0404	4	16	100.000	1.119	572	12
d0406	4	24	100.000	13.048	6.598	18
d0804	8	16	100.000	139.364	72.621	22

Sur une plage de valeurs aussi faible, il est inévitable que chaque valeur soit représentée plusieurs fois. En raison du nombre de tuples (voir tableau 6.5), ceci augmente d'autant la probabilité d'avoir des enregistrements dont les champs ont des valeurs identiques. En consé-

Table 6.6 – Extrait de la table d0404

ID	Attr1	Attr2	Attr3	Attr4
1	106	21	62	88
2	31	13	110	47
3	4	101	124	83
4	82	54	96	26
5	63	23	18	93
6	5	32	9	117
7	91	107	62	56
8	122	14	42	0

quence, les extensions de résumés sont plus fournies, ce qui permet de vérifier l'efficacité du filtrage (voir la section 6.1.1).

De telles données permettent de placer la structure d'index dans la situation qui lui est la moins favorable vis-à-vis des données. En effet, le nombre maximal de feuilles dans un arbre de résumés dépend du nombre de descripteurs dans la base de connaissances et du nombre de dimensions (voir [116]). Mais puisque les cellules vides ne sont pas représentées dans l'arbre, le nombre réel de feuilles dans une hiérarchie constituée pour un jeu de données dépend du placement de ces données par rapport aux variables linguistiques. Des valeurs aléatoires dispersées sur tout le domaine pour chaque attribut font de l'espace indexé un espace plein. Dans le cas de données réelles, l'espace serait plutôt creux. À titre d'exemple, le jeu de données du CIO présente un espace occupé à 2%.

Avant de pouvoir exécuter des requêtes gérées par la méthode d'accès SAINTETIQ, il est nécessaire de créer un index de ce type pour chaque table. Le tableau 6.7 montre l'instruction correspondante pour la table « d0404 ».

Table 6.7 – Création de l'index SAINTETIQ

```
CREATE INDEX seq_d0404
ON d0404
USING saintetiq (attr1, attr2, attr3, attr4, id);
```

6.5.2 Requêtes et résultats

Le jeu de requêtes utilisé couvre l'ensemble des attributs pour chaque jeu de données aléatoires. Les enregistrements identifiés selon le jeu de données interrogé sont les mêmes. En effet, les enregistrements des tables à 4 attributs reprennent les valeurs des quatre premiers champs des enregistrements à 8 attributs.

Le nombre d'attributs des requêtes varie de 1 à la dimension de la table. Pour garantir un résultat non vide, les requêtes de test sont construites à partir d'un enregistrement aléatoirement choisi. Les critères de sélection des requêtes sont des égalités des attributs avec leur valeur dans l'enregistrement. Garantir que les requêtes ont un résultat non-vide permet que l'exploration des résumés ne sera pas arrêtée rapidement en raison d'une détection précoce de l'absence de résultat. Par exemple, les requêtes suivantes (où « `[Table]` » sera instancié respectivement par « `d0404` », « `d0406` » et « `d0804` ») peuvent être dérivées de l'enregistrement 1 du tableau 6.6 :

- `SELECT * FROM [Table] WHERE Attr1 = 106;`
- `SELECT * FROM [Table] WHERE Attr2 = 21;`
- `SELECT * FROM [Table] WHERE Attr3 = 62;`
- `SELECT * FROM [Table] WHERE Attr4 = 88;`
- `SELECT * FROM [Table] WHERE Attr1 = 106 AND Attr2 = 21;`
- `SELECT * FROM [Table] WHERE Attr1 = 106 AND Attr3 = 62;`
- `...`
- `SELECT * FROM [Table] WHERE Attr1 = 106 AND Attr2 = 21 AND Attr3 = 62 AND Attr4 = 88;`

Les requêtes sont générées aléatoirement en reprenant le schéma d'énumération déjà utilisé en section 4.1.2. D'abord, un enregistrement est tiré au sort parmi les 100.000 enregistrements du jeu de données. L'énumération de ses attributs permet d'énumérer toutes les requêtes ayant ses valeurs d'attribut comme critères. Suivant la dimension de la table de données, cet enregistrement permet de composer 15 (table à 4 attributs) ou 255 (8 attributs) requêtes. Cette procédure de tirage aléatoire et de génération exhaustive des requêtes permet d'obtenir deux jeux de requêtes, Q04 et Q08, avec respectivement 5.336 et 5.120 requêtes, utilisés pour interroger les tables appropriées (d0404 et d0406 pour Q04, et d0804 pour Q08).

Les tableaux 6.8 à 6.11 présentent les résultats obtenus avec une taille de buffer de 8 Ko. Une entrée/sortie disque équivaut à un chargement du buffer. Les colonnes des tableaux représentent :

- le nombre d'attributs dans la requête SQL ;

- le temps mis par l'index pour transformer les critères SQL sous la forme exposée en section 2.1.1, explorer la structure d'index et fournir au SGBD tous les tuples de la réponse ;
- le nombre de résumés soumis aux tests d'appariement (section 2.2.2) ; il est équivalent au nombre de nœuds visités ;
- le nombre d'enregistrements répondant à la requête ;
- le volume de l'extension des résumés qui subissent l'étape de filtrage (voir section 6.1.1) ;
- le nombre d'accès disque effectués.

Les tableaux 6.12 à 6.14 présentent les statistiques obtenues en exécutant la totalité des jeux de requêtes. Ils indiquent pour chaque nombre d'attributs dans la requête SQL :

- le nombre de requêtes présentant ce nombre d'attributs ;
- la valeur minimale, la valeur maximale, la moyenne et l'écart-type des accès disque ;
- la valeur minimale, la valeur maximale, la moyenne et l'écart-type des temps d'exécution.

Table 6.8 – Résultats pour 'CIO'

Attributs	Temps	Appariements	Résultats	Tuples filtrés	Nombre d'E/S
1	120 ms	9393 (34,4%)	373	7338	439
1	80 ms	6118 (24,4%)	284	5363	355
2	30 ms	3063 (11,2%)	79	1695	266
3	20 ms	3063 (11,2%)	1	1695	266
4+	10 ms	3063 (11,2%)	1	1695	266

6.5.3 Analyse

Les tableaux de résultats montrent une décroissance très nette du temps de réponse moyen dès que le nombre d'attributs est supérieur à 1. Ceci s'explique par le fait que les index multi-critères ne peuvent pas être groupés pour un nombre de critères inférieur à la dimension gérée (les index dits « groupés » sont discutés en section 5.2.4). Le phénomène est plus marqué lorsqu'il n'y a qu'un critère. Les résultats sont alors répartis dans la plupart des feuilles de l'arbre : près de la moitié des nœuds de l'index est testée pour les jeux de données aléatoires. Il est également corroboré par le fait que le nombre d'appariements effectués décroît lorsque la dimension augmente.

On note que le nombre de tuples résultats décroît lorsque la dimension croît, ce qui est en adéquation avec le fait que chaque dimension équivaut à une contrainte supplémentaire sur

Table 6.9 – Résultats pour 'd0404'

Attributs	Temps	Appariements	Résultats	Tuples filtrés	Nombre d'E/S
1	791 ms	461 (41,2%)	753	33589	206
	621 ms	365 (32,6%)	748	31637	201
2	81 ms	461 (41,2%)	753	33589	206
	10 ms	204 (18,2%)	6	7785	86
3	81 ms	461 (41,2%)	753	33589	206
	0 ms	103 (9,2%)	1	2081	40
4	11 ms	55 (4,9%)	1	523	27
	0 ms	35 (3,1%)	1	482	16

Table 6.10 – Résultats pour 'd0406'

Attributs	Temps	Appariements	Résultats	Tuples filtrés	Nombre d'E/S
1	1042 ms	4437 (34%)	772	39122	457
	751 ms	3529 (27%)	746	34822	437
2	140 ms	3880 (29,7%)	4	38754	412
	10 ms	953 (7,3%)	5	6670	136
3	101 ms	4437 (34%)	1	39122	457
	0 ms	371 (2,8%)	1	939	67
4	11 ms	156 (1,2%)	1	205	54
	0 ms	36 (0,3%)	1	178	11

les données. Ce fait est également cohérent avec les projections d'espaces sur des dimensions inférieures, également discutées en section 5.2.4. De même, le nombre d'appariements décroît avec la dimension de la requête, ce qui correspond à un report des mêmes restrictions sur les résumés qui encapsulent les données.

On note également que le nombre de tuples filtrés est élevé par rapport au nombre de résultats. La partition des domaines par les variables linguistiques puis la réécriture des valeurs précises des critères SQL en modalités plus grossières, expliquent ce fait. Néanmoins, une partition plus fine semble ne pas suffire à augmenter la sélectivité de l'étape de filtrage. Les tuples résultats pour les tableaux 6.9 et 6.10 sont les mêmes, mais si le nombre de tuples filtrés passe

Table 6.11 – Résultats pour 'd0804'

Attributs	Temps	Appariements	Résultats	Tuples filtrés	Nombre d'E/S
1	1462 ms	51338 (36,8%)	787	44259	1888
	1101 ms	44996 (32,3%)	771	43788	1437
2	331 ms	46877 (33,6%)	8	42692	1724
	70 ms	14961 (10,7%)	10	9912	724
3	331 ms	46877 (33,6%)	1	42692	1724
	20 ms	4802 (3,4,%)	1	2442	334
4	331 ms	46877 (33,6%)	1	42692	1724
	10 ms	3553 (2,6%)	1	632	448
5	641 ms	46877 (33,6%)	1	42692	1724
	0 ms	1526 (1,1%)	1	143	252
6	361 ms	47022 (33,7%)	1	41496	1649
	0 ms	1576 (1,1%)	1	151	294
7	291 ms	51338 (36,8%)	1	44259	1888
	0 ms	212 (0,15%)	1	10	58
8	10 ms	424 (0,3%)	1	6	120
	0 ms	141 (0,1%)	1	7	51

Table 6.12 – Statistiques pour 'd0404'

Attr.	Req.	E/S				Temps d'exécution			
		Min	Max	Moyenne	Écart-type	Min	Max	Moyenne	Écart-type
1	667	201	216	207,19	5,74	60	791	685,19	87,66
2	2001	62	216	163,74	61,93	10	81	52,58	24,73
3	2001	22	216	107,60	72,68	0	81	28,92	26,31
4	667	10	40	22,28	6,03	0	11	1,44	3,52
Total	5336	10	216	130,43	78,98	0	791	116,39	218,97

Table 6.13 – Statistiques pour 'd0406'

Attr.	Req.	E/S				Temps d'exécution			
		Min	Max	Moyenne	Écart-type	Min	Max	Moyenne	Écart-type
1	667	358	457	420,51	32,57	80	1042	861,25	101,14
2	2001	87	457	337,71	122,74	10	140	67,15	35,27
3	2001	23	457	224,86	147,47	0	101	35,77	36,29
4	667	10	103	46,15	17,33	0	11	1,59	3,66
Total	5336	10	457	269,29	159,84	0	1042	146,45	275,13

Table 6.14 – Statistiques pour 'd0804'

Attr.	Req.	E/S				Temps d'exécution			
		Min	Max	Moyenne	Écart-type	Min	Max	Moyenne	Écart-type
1	40	1437	1888	1683,23	154,80	260	1462	1082,53	204,34
2	280	707	1888	1577,31	300,10	70	331	248,46	66,16
3	840	334	1888	1454,07	397,81	20	331	213,93	86,71
4	1400	197	1888	1307,23	475,38	10	331	183,03	101,51
5	1400	128	1888	1127,15	534,79	0	641	151,79	111,88
6	840	87	1888	897,46	565,78	0	361	107,61	107,02
7	280	42	1888	582,40	532,53	0	291	61,14	92,00
8	40	37	147	82,20	24,79	0	10	1,25	3,35
Total	5120	37	1888	1183,35	555,92	0,00	1462,00	169,70	138,46

du simple au triple pour 4 attributs, il varie à peine dans le tableau 6.10 pour 2 attributs, et est même plus important pour 1 attribut alors que la partition plus fine laisserait supposer le contraire.

Le nombre important d'accès disque s'explique, en partie, par la structure du fichier d'index. Lorsque le test d'un nœud révèle que des résultats pourraient être trouvés dans le sous-arbre, il est certain que chacun des nœuds fils sera également testé. Dans un parcours préfixé d'un arbre, seul le parent et le premier fils se suivent (cf. figure 6.3). Ceci impose parfois un accès disque supplémentaire pour chaque autre fils, éventuellement infructueux si le fils ne présente aucune

correspondance. Cependant, la taille de la structure d'un nœud interne est relativement faible (grâce à l'utilisation des indices pour représenter l'intension – voir section 6.2) et épargne un certain nombre d'accès disque lorsque le parcours atteint les niveaux les plus bas de l'arbre. À ces niveaux, une entrée est de taille minimale et les blocs physiques lus contiennent plus d'entrées de résumés. En conclusion, le nombre d'E/S suggère une autre organisation du fichier d'index. Celle-ci consisterait à reporter l'intension des nœuds fils au sein de l'entrée du nœud parent. Rappelons que les tests d'appariement n'utilisent que l'intension du résumé. En rendant disponibles en un même accès disque toutes les intensions qui seront testées, on peut escompter une diminution notable du nombre d'E/S.

On peut remarquer que la structure du fichier reproduit la structure logique issue de la classification qu'opère le processus de résumé. L'équivalence entre nœud physique et nœud logique qu'on peut observer dans les index classiques n'est pas reproduite ici. Les accès disque n'étant pas optimisés pour tenir compte de cette équivalence, leur nombre est élevé.

Toujours à propos du nombre d'accès disque, le passage d'une version XML vers un fichier binaire avait pour but de compacter la hiérarchie de résumés représentée. Il est certain qu'une modification des types de données utilisés permettrait de réaliser un meilleur compactage du fichier. Par exemple, le nombre moyen de fils d'un résumé est inférieur à cinq pour les jeux de données du tableau 6.5, et le nombre maximal de fils d'un nœud est borné par le cardinal maximal d'un domaine réécrit de la base de connaissances (voir chapitre 1 pour la définition d'un domaine réécrit). Par conséquent, un type BYTE (1 octet) plutôt que INT4 (4 octets) fait gagner 75% du volume sur ce seul champ du nombre de fils ; ce qui, rapporté au nombre de résumés dans une hiérarchie, constitue un volume non négligeable, la même opération de changement de type pouvant être répétée pour d'autres champs du fichier.

Enfin, une part des accès disque vient du fait que le terme d'une recherche peut intervenir tardivement dans le parcours de la hiérarchie de résumés, car une réponse vide ne peut être déterminée qu'après la phase de filtrage. Notons que le filtrage requiert l'examen de chaque enregistrement et probablement des lectures supplémentaires, mais celles-ci ne sont pas décomptées dans les tableaux 6.9 à 6.11 de résultats. En effet, l'accès aux tables de données étant géré par le SGBD, l'impact du filtrage se fait ressentir sur les temps de réponse, mais pas sur le nombre d'accès disque présenté.

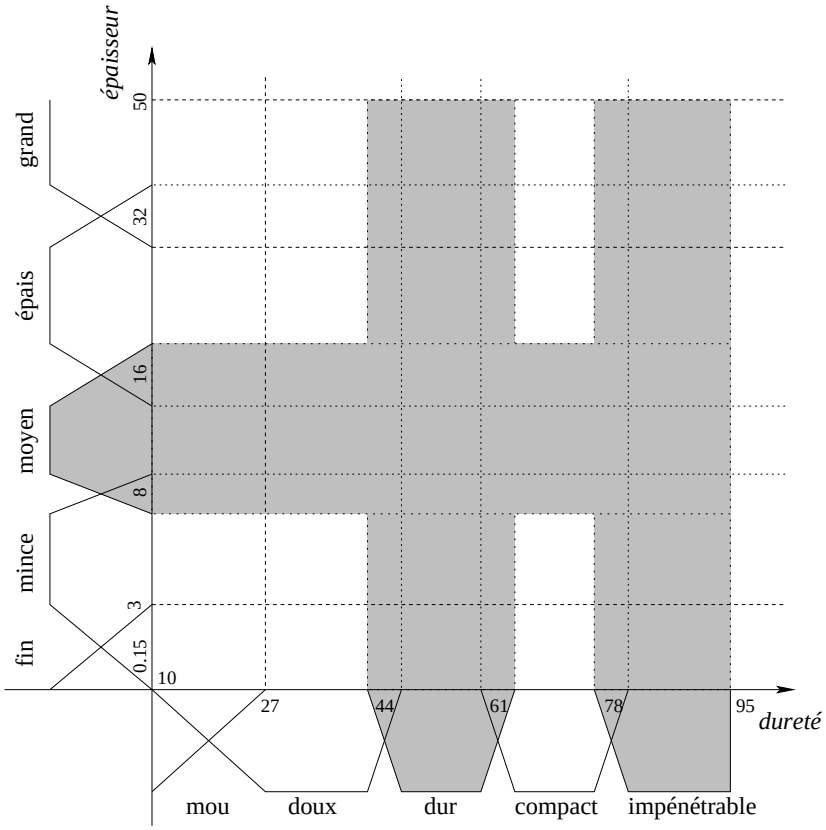


Figure 6.4 – Désactivation de descripteurs

6.6 Conclusion et perspectives

Nous avons présenté une utilisation de résumés SAINTETIQ en tant que structure d'index intégrée au SGBD PostgreSQL. En exploitant des hiérarchies de résumés déjà construites, l'objectif est d'estimer leur efficacité pour une recherche dans un index. Cet aspect de l'interrogation est largement différent de celui étudié jusqu'ici, qui consistait à caractériser les données par des descriptions généralisées.

En dehors des requêtes monocritères, une hiérarchie des résumés, bien que conçue avec un autre objectif que celui d'obtenir un index et non optimisée dans ce but, fournit des temps de réponse acceptables pour un prototype. Il reste cependant difficile d'expliquer ces temps de réponse en les rattachant à un ou plusieurs facteurs, par exemple la finesse des partitions de domaines. La perspective immédiate est d'utiliser des volumes de données plus importants afin de marquer des différences. Mais les données réelles sont difficiles à trouver. Nous travaillons donc à la génération de données aléatoires avec pour contrainte d'atteindre un taux cible d'es-

pace creux, ceci afin de simuler des données réelles. Ce taux correspond à la proportion des cellules non vides par rapport au nombre total de cellules. Dans les données réelles, cette proportion est faible : 2% pour le jeu de données bancaires.

Pour l'heure, un taux fixé (par exemple, 10% sur un jeu de données à 8 dimensions) est obtenu en « désactivant » certains descripteurs. La désactivation consiste à rejeter toute valeur aléatoirement générée qui serait décrite par le descripteur désactivé, permettant ainsi de réduire le nombre des cellules contenant des données. La figure 6.4 offre un point de comparaison avec la figure 5.16 (en page 122) en montrant, en grisé, les cellules qui seront vides. Sur cet exemple à 2 dimensions, le nombre de cellules est de 12 contre 25 sans désactivation. Une fois que des hypothèses auront été formulées et confirmées quant aux facteurs de performance, il sera possible d'adapter la construction des résumés dans l'optique d'une utilisation effective en tant que structure d'index. Entreront alors en ligne de compte les opérations de mises à jour proposées dans [125].

Conclusion générale

Résumé

Durant cette thèse, nous avons étudié les résumés linguistiques du modèle SAINTETIQ sous l'angle de l'interrogation. Le modèle SAINTETIQ, proposé par Raschia et Mouaddib [116], construit une hiérarchie de résumés d'une relation R de données structurées. Chaque résumé est une description concise d'un sous-ensemble des données relationnelles de R. Notre but a été d'offrir les moyens de trouver, au sein d'une hiérarchie de résumés, les résumés ou les données satisfaisant des critères spécifiés. Nous nous sommes également attelé à enrichir l'interrogation des résumés de fonctionnalités qui semblaient intéressantes, et à lui trouver des domaines d'application.

Le premier chapitre a permis de présenter les résumés du modèle SAINTETIQ. De cette présentation, il faut retenir les propriétés autour desquelles la contribution du présent document est bâtie. Il s'agit en premier lieu de l'organisation en arbre des hiérarchies de résumés. De cette organisation, découle un ordre partiel entre les nœuds de l'arbre (c'est-à-dire les résumés) dont la sémantique est celle d'une couverture, d'un "englobement". Ainsi, les données décrites par un nœud de niveau inférieur, par exemple une feuille de l'arbre, sont couvertes par tout parent de ce nœud. Les mêmes données sont donc décrites par plusieurs résumés de granularités différentes. Outre l'organisation de l'arbre, la propriété d'unicité d'un résumé dans une hiérarchie est à mentionner. Elle permet d'exclure toute redondance autre que celle engendrée par le recouvrement des étiquettes linguistiques. Aucune ambigüité ne peut ainsi survenir quant à la représentation d'un résumé, que ce soit en intension ou en extension. De fait, l'unicité facilite les recherches au sein d'une hiérarchie de résumés : elle garantit que tout résumé correspondant à une expression est le seul à fournir cette adéquation. Bien sûr, cette garantie ne vaut que pour les expressions spécifiées sur tous les attributs.

Dans le chapitre suivant, une procédure d'exploration des hiérarchies de résumés est proposée. Dans cette procédure, nous faisons l'hypothèse qu'une recherche est menée dans la hiérarchie afin de trouver les résumés qui satisfont des critères dits « de sélection ». La recherche fait suite à une requête et fournit des résumés en résultat. La requête, comprenant tous les critères de sélection, adopte la forme d'un ensemble de descripteurs linguistiques, regroupés par attribut.

Le chapitre 3 a présenté un état de l'art de l'interrogation flexible en bases de données, avec le souci de caractériser les systèmes qui entrent dans ce cadre. Pour ce faire, la littérature a servi de source d'inspiration. Les différentes classifications rencontrées ont été rassemblées et constituent une proposition de définition des systèmes d'interrogation flexible en bases de données. Outre cette définition, l'omniprésence de la notion de « préférence » a été mise en évidence. Les préférences ont donc été étudiées sous l'angle de leur intégration dans des systèmes existants, révélant ainsi la notion de tri des résultats de l'interrogation en fonction de leur adéquation à la requête. Les traits communs des préférences ont été exposés, ainsi que les caractéristiques propres à chaque élément de la typologie utilisée, reprise de la littérature.

Deux types de systèmes ont été distingués suivant leurs fondements théoriques : les systèmes basés sur la logique du premier ordre et ceux également basés sur la théorie des sous-ensembles flous. Les premiers réalisent une extension du langage SQL par l'adjonction de clauses de préférences. Les seconds reprennent la même base théorique à leur compte puisqu'ils sont tous orientés vers le langage SQL. Ils y rajoutent les sous-ensembles flous pour modéliser le caractère graduel de conditions de sélection. Une adéquation graduelle des résultats découle naturellement des conditions ainsi formées. L'objectif n'étant pas d'évaluer les avantages et inconvénients des divers systèmes, une étude comparative n'a pas été effectuée. Cependant, cet état de l'art récapitulatif se veut un point de départ pour une telle étude ou d'autres travaux dans le champ de l'interrogation flexible.

L'interrogation flexible sert de cadre à un ensemble des propositions exposées dans le chapitre 4. On remarque que les résumés linguistiques du modèle SAINTETIQ partagent avec certains travaux en interrogation flexible le recours aux sous-ensembles flous pour modéliser le caractère graduel de l'appartenance à un ensemble ou de la satisfaction d'une condition. Ceci ne semble pas suffisant pour trouver une application aux résumés linguistiques de données structurées dans ces systèmes. Néanmoins, un modèle d'interrogation approchée des données relationnelles a été proposé et implémenté [141, 143]. Les principes d'une modification des requêtes ont été posés, dans le but d'enrichir le modèle d'un comportement coopératif [140, 142]. Enfin, l'utilisation d'un vocabulaire distinct du vocabulaire de construction des résumés a été envisagée [137, 138]. Ce dernier travail a été doublement motivé : d'une part, par la popularité du domaine d'application de la personnalisation et d'autre part, par la nécessité de gérer la possibilité (offerte par certains systèmes) de définir de nouveaux termes.

Notre volonté d'offrir aux résumés linguistiques de SAINTETIQ un débouché en interrogation flexible nous a conduit à étudier la possibilité de fournir des informations sur les tuples décrits par les résumés. La présentation des enregistrements résultats en fonction de leur per-

tinence vis-à-vis de la requête traitée, caractéristique des systèmes d'interrogation flexible en bases de données, ne peut se satisfaire de résumés linguistiques. Ces derniers offrant des références aux enregistrements qu'ils décrivent, l'utilisation d'une hiérarchie de résumés comme structure d'index a été envisagée. Le chapitre 5 débute cette étude par un tour d'horizon des index, orienté principalement vers les index multidimensionnels. Cet état de l'art, en plus d'avoir exposé une sélection de techniques d'indexation, nous a permis d'aboutir à la conclusion que l'utilisation des hiérarchies de résumés comme index était plausible.

Afin de vérifier cette conclusion et poursuivre l'étude, un travail expérimental a été entrepris. Il a consisté à intégrer des hiérarchies de résumés au sein d'un système de gestion de bases de données opérationnel. Il a ainsi été possible d'exécuter des requêtes de sélection utilisant des résumés comme méthode d'accès. La validité de l'approche est démontrée, mais d'autres travaux restent nécessaires pour déterminer les facteurs qui influent sur son efficacité et éventuellement, l'améliorer.

La contribution de cette thèse se retrouve également dans les applications offertes au travers de prototypes qui donnent vie à certaines de nos propositions.

Perspectives

Le travail de thèse présenté dans ce document touche à plusieurs champs d'étude qui peuvent être enrichis ou étudiés plus en détail. Ainsi, l'algorithme présenté au chapitre 2 peut être généralisé dans ses principes, aussi bien dans la formulation des requêtes que dans la présentation des résultats. Par exemple, on peut envisager une forme canonique des requêtes exprimée non pas en forme normale conjonctive, mais en logique du premier ordre. Les critères ne seraient plus des conditions logiques, mais des prédicats. Dès lors, les deux facettes de l'interrogation des résumés (le test d'existence et la description des données) seraient formalisées de façon plus rigoureuse. De plus, l'utilisation de prédicats comme critères de sélection est plus en rapport avec l'usage courant (dans le langage SQL).

L'exploitation, lors de l'interrogation de hiérarchies de résumés, des degrés et mesures affectés aux résumés constitue la perspective la plus intéressante. Par la suite, on peut légitimement supposer que ces enrichissements affecteront la suite du processus d'interrogation, par exemple, la présentation des résultats.

D'autre part, la personnalisation est un domaine que nous avons à peine abordé. Du fait de l'aspect descriptif des résumés, interroger une hiérarchie permet, dans certaines circonstances,

de déterminer très rapidement l'absence de résultats. En matière de personnalisation, une application évidente est la sélection des sources de données dans un contexte multisources. D'autres applications pour les résumés linguistiques restent probablement à découvrir.

Enfin, l'intégration des hiérarchies de résumés dans un SGBD est une tâche difficile, mais qui reste un objectif attrayant. La poursuite de ce travail permettrait d'identifier une organisation adaptée des hiérarchies en tant qu'index ayant une base dans le flou. L'expérience acquise grâce à la première implémentation présentée dans ce document devrait rendre cette poursuite relativement aisée. Nous pensons qu'il suffira de jeux de données plus conséquents et d'un meilleur protocole d'expérimentation, bien défini quant aux éléments de performance et d'utilité évalués. Notre objectif initial de couplage entre les résumés linguistiques du modèle SAINTETIQ et les systèmes d'interrogation flexible ne sera atteint qu'à cette condition. À terme, cette approche devra être comparée aux méthodes d'évaluation de requêtes flexibles.

Bibliographie

[1] Paul M. AOKI.
Generalizing "search" in generalized search trees (extended abstract).
Dans *Proceedings of the Fourteenth ICDE*, pages 380–389. IEEE Computer Society, février 1998.

[2] Brian BABCOCK, Shivnath BABU, Mayur DATAR, Rajeev MOTWANI et Jennifer WIDOM.
Models and issues in data stream systems.
Dans *Proceedings of the 21st PODS*, pages 1–16. ACM, juin 2002.

[3] Daniel BARBARÁ, William DUMOUCHEL, Christos FALOUTSOS, Peter J. HAAS, Joseph M. HELLERSTEIN, Yannis E. IOANNIDIS, H. V. JAGADISH, Theodore JOHNSON, Raymond T. NG, Viswanath POOSALA, Kenneth A. ROSS et Kenneth C. SEVCIK.
The New Jersey Data Reduction Report.
IEEE Data Engineering Bulletin, 20(4):3–45, 1997.

[4] Rudolf BAYER.
The universal B-Tree for multidimensional indexing.
Dans *Proceedings of WWCA'97 Worldwide Computing and Its Applications, International Conference, Tsukuba, Japan*, volume 1274 de *LNCS*, pages 198–209, mars 1997.

[5] Rudolf BAYER et Edward M. MCCREIGHT.
Organization and maintenance of large ordered indices.
Acta Informatica, 1(3):173–189, 1972.

[6] Norbert BECKMANN, Hans-Peter KRIEGEL, Ralf SCHNEIDER et Bernhard SEEGER.
The R*-Tree: an efficient and robust access method for points and rectangles.
Dans *Proceedings of ACM SIGMOD International Conference on Management of Data, Atlantic City, New Jersey*, pages 322–331, mai 1990.

[7] Mihir BELLARE et Daniele MICCIANCIO.
A new paradigm for collision-free hashing: incrementality at reduced cost.
Cryptology ePrint Archive, Report 1997/001, 1997.
Disponible à l'adresse
http://eprint.iacr.org/.

[8] Jon Louis BENTLEY.
Multidimensional binary search trees used for associative searching.
Communications of the ACM, 18(9):509–517, 1975.

[9] Jon Louis BENTLEY et Jerome H. FRIEDMAN.
Data structures for range searching.
ACM Computing Surveys, 11(4):397–409, 1979.

[10] Stefan BERCHTOLD, Christian BÖHM, H. V. JAGADISH, Hans-Peter KRIEGEL et Jörg SANDER.
Independent quantization: an index compression technique for high-dimensional data spaces.
Dans *Proceedings of ICDE*, pages 577–588, 2000.

[11] Stefan BERCHTOLD, Christian BÖHM, Daniel A. KEIM et Hans-Peter KRIEGEL.
A cost model for nearest neighbor search in high-dimensional data space.
Dans *Proceedings of PODS, Tucson, Arizona*, pages 78–86. ACM Press, mai 1997.

[12] Stefan BERCHTOLD, Christian BÖHM et Hans-Peter KRIEGEL.
The Pyramid-Technique: towards breaking the curse of dimensionality.
Dans *Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington*, pages 142–153, juin 1998.

[13] Stefan BERCHTOLD, Daniel A. KEIM et Hans-Peter KRIEGEL.
The X-Tree: an index structure for high-dimensional data.
Dans *Proceedings of 22nd VLDB, Mumbai, India*, pages 28–39, 1996.

[14] Sid-Ahmed BERRANI, Laurent AMSALEG et Patrick GROS.
Recherche par similarité dans les bases de données multidimensionnelles : panorama des techniques d'indexation.
RSTI - Ingénierie des systèmes d'information. Bases de données et multimédia, 7(5-6):9–44, 2002.

[15] Alain BIDAULT.
Affinement de requêtes posées à un médiateur.
Thèse de doctorat, Université de Paris XI Orsay, juillet 2002.

[16] Alain BIDAULT, Christine FROIDEVAUX et Brigitte SAFAR.
Finding successful queries in a mediator context.
Dans *Proceedings of FQAS 2000*, pages 171–181, octobre 2000.

[17] Alain BIDAULT, Christine FROIDEVAUX et Brigitte SAFAR.
Repairing queries in a mediator approach.

Dans *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 406–410, 2000.

[18] Alain BIDAULT, Christine FROIDEVAUX et Brigitte SAFAR.
Similarity between queries in a mediator.
Dans *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 235–239, 2002.

[19] Ignacio BLANCO, Nicolás MARÍN, Olga PONS et María Amparo Vila MIRANDA.
An extension of data description language (DDL) for fuzzy data handling.
Dans *Proceedings of the FQAS Int. Conf.*, pages 54–63, 2000.

[20] Christian BÖHM, Stefan BERCHTOLD, Hans-Peter KRIEGEL et Urs MICHEL.
Multidimensional index structures in relational databases.
Journal of Intelligent Information Systems, 15:51–70, 2000.

[21] Patrick BOSC, Didier DUBOIS et Henri PRADE.
Fuzzy functional dependencies and redundancy elimination.
JASIS, 49(3):217–235, 1998.

[22] Patrick BOSC, Ludovic LIÉTARD, Olivier PIVERT et Daniel ROCACHER.
Gradualité et imprécision dans les bases de données.
Ellipses, Paris, 2004.

[23] Patrick BOSC, Amihai MOTRO et Gabriella PASI.
Report on the fourth international conference on flexible query answering systems.
SIGMOD Record, 30(1):66–69, 2001.

[24] Patrick BOSC et Olivier PIVERT.
Fuzzy queries and relational databases.
Dans *Proceedings of the ACM Symposium on Applied Computing*, pages 170–174, Phoenix, AZ, USA, mars 1994.

[25] Patrick BOSC et Olivier PIVERT.
SQLf: a relational database language for fuzzy querying.
IEEE Transactions on Fuzzy Systems, 3(1):1–17, 1995.

[26] Patrick BOSC, Olivier PIVERT et Laurent UGHETTO.
On data summaries based on gradual rules.
Dans *Fuzzy Days*, pages 512–521, 1999.

[27] Patrick BOSC et Henri PRADE.
An introduction to the fuzzy set and possibility theory-based treatment of flexible queries and uncertain or imprecise databases.

Dans *Uncertainty Management in Information Systems*, pages 285–324. Kluwer Academic Publishers, Boston, 1996.

Disponible à l'adresse
citeseer.nj.nec.com/bosc94introduction.html.

[28] Bernadette BOUCHON-MEUNIER.
La logique floue.
Presses Universitaires de France, 3è édition, novembre 1999.

[29] Jean-Marie BOUROCHE et Gilbert SAPORTA.
L'analyse des données.
Presses Universitaires de France, Paris, 1980.

[30] Tolga BOZKAYA et Z. Meral ÖZSOYOGLU.
Distance-based indexing for high-dimensional metric spaces.
Dans Joan PECKHAM, réd., *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 357–368. ACM Press, 1997.

[31] Ronen I. BRAFMAN et Carmel DOMSHLAK.
Database Preference Queries Revisited.
Technical Report TR2004-1934, Cornell University Computing and Information Science, 2004.

[32] Gilles CELEUX, Edwin DIDAY, Gérard GOVAERT, Yves LECHEVALLIER et H. RALAMBONDRAINY.
Classification automatique des données - aspects statistiques et informatiques.
Dunod Informatique. Bordas, Paris, 1989.

[33] Kaushik CHAKRABARTI et Sharad MEHROTRA.
The Hybrid Tree: an index structure for high dimensional feature spaces.
Dans *ICDE*, pages 440–447, 1999.

[34] Jan CHOMICKI.
Querying with intrinsic preferences.
Dans *Proceedings of EDBT 2002, Prague, Czech Republic*, pages 34–51, mars 2002.

[35] Jan CHOMICKI.
Preference formulas in relational queries.
ACM Transactions on Database Systems, 28(4):427–466, 2003.

[36] Wesley W. CHU, Hua YANG, Kuorong CHIANG, Michael MINOCK, Gladys CHOW et Chris LARSON.

CoBase: a scalable and extensible cooperative information system.
Journal of Intelligent Information Systems, 6(2/3):223–259, 1996.
Disponible à l'adresse
citeseer.ist.psu.edu/chu96cobase.html.

[37] Paolo CIACCIA, Marco PATELLA et Pavel ZEZULA.
M-Tree: an efficient access method for similarity search in metric spaces.
Dans *VLDB'97, Proceedings of 23rd VLDB, Athens, Greece*, pages 426–435, août 1997.

[38] Philippe CIBOIS.
L'analyse factorielle.
Presses Universitaires de France, Paris, 1983.

[39] William G. COCHRAN.
Sampling techniques.
Wiley & Sons, New York, 1977.

[40] Edgar F. CODD.
A relational model for large shared data banks.
Communications of the ACM, 13(6):377–387, juin 1970.

[41] Edgar F. CODD.
Relational completeness of data base sublanguages.
Dans Randall J. RUSTIN, réd., *Data base Systems*, pages 65–98. Prentice Hall, 1972.
IBM Research Report RJ987.

[42] Edgar F. CODD.
Providing OLAP (On-Line Analytical Processing) to useranalysts: an IT mandate.
Dans *Technical Reports*, 1993.

[43] Douglas COMER.
The ubiquitous B-Tree.
ACM Computing Surveys, 11(2):121–137, 1979.

[44] James W. COOLEY et John W. TUKEY.
An algorithm for the machine calculation of complex Fourier series.
Math. Comput, 19:297–301, 1965.

[45] Antoine CORNUÉJOLS, Laurent MICLET et Yves KODRATOFF.
Apprentissage artificiel: concepts et algorithmes.
Eyrolles, Paris, 2002.

[46] Juan C. CUBERO, Juan Miguel MEDINA, Olga PONS et María Amparo Vila MIRANDA.
Data summarization in relational databases through fuzzy dependencies.

Information Sciences, 121(3-4):233–270, 1999.

[47] Frédéric CUPPENS et Robert DEMOLOMBE.
Cooperative answering: a methodology to provide intelligent access to databases.
Dans *Proceedings of the 2nd International Conference on Expert Database Systems*, pages 621–643, 1988.

[48] Chris J. DATE.
An introduction to database systems.
Addison-Wesley, 6è édition, novembre 1994.

[49] Chris J. DATE.
Introduction aux bases de données.
Ed. Vuibert Informatique, 7è édition, décembre 2000.

[50] Robert DEMOLOMBE.
Abstract objects to represent large answers to queries in a concise form.
Dans *Proceedings of FQAS 2000*, pages 171–181, octobre 2000.

[51] Didier DUBOIS et Henri PRADE.
Using fuzzy sets in database systems: why and how ?
Dans *Proceedings of the 1996 Workshop on Flexible Query-Answering Systems*, pages 89–103, Roskilde, Denmark, mai 1996.

[52] Didier DUBOIS et Henri PRADE.
The three semantics of fuzzy sets.
Fuzzy Sets and Systems, 90:141–150, 1997.

[53] Didier DUBOIS et Henri PRADE.
Fuzzy sets in data summaries — outline of a new approach.
Dans *Proc. 8th International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'2000)*, volume 2, pages 1035–1040, juillet 2000.

[54] Pierre DUHAMEL et Martin VETTERLI.
Fast Fourier Transforms: a tutorial review and a state of the art.
Signal Processing, 19(4):259–299, 1990.

[55] Brians S. EVERITT, Sabine LANDAU et Morven LEESE.
Cluster analysis.
Hodder Arnold, London, 4è édition, 2001.

[56] Ronald FAGIN, Jurg NIEVERGELT, Nicholas PIPPENGER et H. Raymond STRONG.
Extendible hashing – a fast access method for dynamic files.

ACM TODS, 4(3):315–344, septembre 1979.

[57] Li FAN, Pei CAO, Jussara ALMEIDA et Andrei Z. BRODER.
Summary cache: a scalable wide-area Web cache sharing protocol.
IEEE/ACM Transactions on Networking, 8(3):281–293, 2000.
Disponible à l'adresse
citeseer.ist.psu.edu/fan98summary.html.

[58] Ephraim FEIG et Shmuel WINOGRAD.
Fast algorithms for the discrete cosine transform.
IEEE Transactions on Signal Processing, 40(9):2174–2193, 1992.

[59] Josef FINK et Alfred KOBSA.
A review and analysis of commercial user modeling servers for personalization on the world wide web.
User Model. User-Adapt. Interact., 10(2-3):209–249, 2000.

[60] Michael FREESTON.
The BANG file: a new kind of grid file.
Dans *Proceedings of ACM SIGMOD*, pages 260–269, 1987.

[61] Michael FREESTON.
A general solution of the n-dimensional B-Tree problem.
Dans *Proceedings of ACM SIGMOD, San Jose, California*, pages 81–90, mai 1995.

[62] Henry FUCHS, Gregory D. ABRAM et Eric D. GRANT.
Near real-time shaded display of rigid objects.
Computer Graphics, 17(3):65–72, juillet 1983.

[63] Henry FUCHS, Zvi M. KEDEM et Bruce F. NAYLOR.
On visible surface generation by a priori tree structures.
ACM Computer Graphics, 14(3):124–133, juillet 1980.

[64] Terry GAASTERLAND, Parke GODFREY et Jack MINKER.
An overview of cooperative answering.
Journal of Intelligent Systems, 1(2):123–157, 1992.

[65] Terry GAASTERLAND, Parke GODFREY, Jack MINKER et Lev NOVIK.
Cooperative answers in database systems.
http://citeseer.ist.psu.edu/50363.html.

[66] Terry GAASTERLAND, Parke GODFREY, Jack MINKER et Lev NOVIK.
A cooperative answering system.
Dans *LPAR*, pages 478–480, 1992.

[67] Terry GAASTERLAND et Jorge LOBO.
Qualified answers that reflect user needs and preferences.
Dans Jorge B. BOCCA, Matthias JARKE et Carlo ZANIOLO, réds., *VLDB'94, Proceedings of 20th VLDB, Santiago de Chile, Chile*, pages 309–320. Morgan Kaufmann, septembre 1994.

[68] Volker GAEDE et Oliver GÜNTHER.
Multidimensional access methods.
ACM Computing Surveys, 30(2):170–231, 1998.

[69] Annie GAL.
Cooperative responses in deductive databases.
Thèse de Doctorat, University of Maryland, College Park, 1988.

[70] José GALINDO, Juan Miguel MEDINA, Olga PONS et Juan C. CUBERO.
A server for fuzzy SQL queries.
Dans *Proceedings of the 3rd International Conference on Flexible Query Answering Systems, Roskilde, Denmark*, volume 1495 de *LNAI*, pages 164–174. Springer, mars 1998.

[71] Didier Le GALL et Ali TABATABAI.
Sub-band coding of digital images using symmetric short kernel filters and arithemetic coding techniques.
Dans *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 761–764, avril 1988.

[72] Anil K. GARG et C. C. GOTLIEB.
Order-preserving key transformations.
ACM TODS, 11(2):213–234, juin 1986.

[73] Diane GREENE.
An implementation and performance analysis of spatial data access methods.
Dans *Proceedings of ICDE*, pages 606–615, 1989.

[74] Herbert Paul GRICE.
Logic and conversation.
Dans *Syntax and Semantics*, volume 3, pages 41–58. Academic Press, 1975.

[75] Herbert Paul GRICE.
Studies in the Way of Words.
Harvard University Press, Cambridge, USA, 1989.

[76] Antonin GUTTMAN.

R-trees: a dynamic index structure for spatial searching.
Dans *Proceedings of ACM SIGMOD International Conference on Management of Data, Boston, Masachussetts*, pages 47–57, juin 1984.

[77] Richard Wesley HAMMING.
Error detecting and error correcting codes.
The Bell System Technical Journal, 27(2):147–160, avril 1950.

[78] Sven Ove HANSSON.
What is ceteris paribus preference.
Journal of Philosophical Logic, 25(3):307–332, 1996.

[79] Joseph M. HELLERSTEIN, Jeffrey F. NAUGHTON et Avi PFEFFER.
Generalized search trees for database systems.
Dans *Proceedings of 21th VLDB*, pages 562–573. Morgan Kaufmann, septembre 1995.

[80] Andreas HENRICH.
The LSD^h-Tree: an access structure for feature vectors.
Dans *Proceedings of the Fourteenth ICDE*, pages 362–369. IEEE Computer Society, février 1998.

[81] Andreas HENRICH, Hans-Werner SIX et Peter WIDMAYER.
The LSD tree: spatial access to multidimensional point and nonpoint objects.
Dans *Proceedings of the 15th VLDB, Amsterdam, The Netherlands*, pages 45–53, août 1989.

[82] Klaus HINRICHS.
Implementation of the Grid File: design concepts and experience.
BIT, 25(4):569–592, 1985.

[83] Vagelis HRISTIDIS, Nick KOUDAS et Yannis PAPAKONSTANTINOU.
PREFER: a system for the efficient execution of multi-parametric ranked queries.
Dans *SIGMOD Conference*, pages 259–270, 2001.
Disponible à l'adresse
citeseer.ist.psu.edu/hristidis01prefer.html.

[84] David A. HUFFMAN.
A method for the construction of minimum-redundancy codes.
Dans *Proceedings of the I.R.E.*, pages 1098–1102, septembre 1952.

[85] Andreas HUTFLESZ, Hans-Werner SIX et Peter WIDMAYER.
Twin Grid Files: space optimizing access schemes.
Dans *Proceedings of ACM SIGMOD*, pages 183–190, juin 1988.

[86] IBM CORPORATION.
OS ISAM Logic, 1966.

[87] Janusz KACPRZYK et Slawomir ZADROŻNY.
Computing with words in intelligent database querying: standalone and Internet-based applications.
Information Sciences, 134:71–109, mai 2001.

[88] Samuel Jerrold KAPLAN.
Cooperative responses from a portable natural language database query system.
Dans M. BRADY, réd., *Computational Models of Discourse*. MIT Press, 1982.

[89] Norio KATAYAMA et Shin'ichi SATOH.
The SR-Tree: an index structure for high-dimensional nearest neighbor queries.
Dans *Proceedings of ACM SIGMOD, Tucson, Arizona*, pages 369–380, mai 1997.

[90] Werner KIESSLING.
Foundations of preferences in database systems.
Dans *Proceedings of 28th VLDB, Hong Kong, China*, pages 311–322, 2002.

[91] Werner KIESSLING et Gerhard KÖSTLER.
Preference SQL - design, implementation, experiences.
Dans *Proceedings of 28th VLDB, Hong Kong, China*, pages 990–1001, 2002.

[92] Donald E. KNUTH.
The Art of Computer Programming. Volume III: Sorting and Searching.
Addison-Wesley, Reading, Massachussetts, 1973.

[93] Hans-Peter KRIEGEL.
Performance comparison of index structures for multi-key retrieval.
Dans *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts*, pages 186–196, juin 1984.

[94] Martial LACROIX et Pierre LAVENCY.
Preferences: putting more knowledge into queries.
Dans *Proceedings of 13th VLDB*, pages 217–225, septembre 1987.

[95] Laks V. S. LAKSHMANAN, Jian PEI et Jiawei HAN.
Quotient Cube: how to summarize the semantics of a data cube.
Dans *Proceedings of VLDB*, pages 778–789, 2002.

[96] Henrik Legind LARSEN.
An approach to flexible information access systems using soft computing.

Dans *Proeedings of the 32nd Hawaii International Conference on System Sciences*, volume 6, janvier 1999.

[97] Do Heon LEE et Myoung Ho KIM.
Database summarization using fuzzy ISA hierarchies.
IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics, 27:68–78, février 1997.

[98] Philip L. LEHMAN et S. Bing YAO.
Efficient locking for concurrent operations on B-Trees.
ACM Transactions on Database Systems, 6(4):650–670, décembre 1981.

[99] Wendy G. LEHNERT.
A computational theory of human question answering.
Dans A. JOSHI, B. WEBBER et I. SAG, réds., *Elements of Discourse Understanding*. Cambridge University Press, 1981.

[100] King-Ip LIN, H. V. JAGADISH et Christos FALOUTSOS.
The TV-Tree: an index structure for high-dimensional data.
VLDB Journal, 3(4):517–542, 1994.

[101] David B. LOMET et Betty SALZBERG.
The hB-Tree: a multiattribute indexing method with good guaranteed performance.
ACM Trans. Database Syst., 15(4):625–658, 1990.

[102] Kathleen R. MCKEOWN.
Generating Natural Language Text in Response to Questions about Database Queries.
Thèse de doctorat, University of Pensylvannia, USA, 1981.

[103] MICROSOFT CORPORATION.
Description of performance options in Windows, Novembre 2006.
Disponible à l'adresse
http://support.microsoft.com/kb/259025/en-us.

[104] Jack MINKER.
Logic and databases: a 20 year retrospective.
Dans *Logic in Databases*, pages 3–57, 1996.
Disponible à l'adresse
citeseer.ist.psu.edu/minker96logic.html.

[105] Jürg NIEVERGELT, Hans HINTERBERGER et Kenneth C. SEVCIK.
The Grid File: an adaptable, symmetric multikey file structure.
ACM Trans. Database Syst., 9(1):38–71, 1984.

[106] Alan V. OPPENHEIM, Ronal W. SCHAFER et John R. BUCK.
Discrete-Time Signal Processing.
Prentice-Hall, New Jersey, 1999.

[107] Jack A. ORENSTEIN.
Spatial query processing in an object-oriented database system.
Dans *Proceedings of ACM SIGMOD International Conference on Management of Data, Washington, D.C.*, pages 326–336, mai 1986.

[108] Jack A. ORENSTEIN et T. H. MERRETT.
A class of data structures for associative searching.
Dans *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Waterloo, Canada*, pages 181–190. ACM, 1984.

[109] Sophocles J. ORFANIDIS.
Introduction to signal processing.
Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[110] Donald B. PERCIVAL et Andrew T. WALDEN.
Wavelet Methods for Time Series Analysis.
Cambridge University Press, Cambridge, UK, 2000.

[111] Henri PRADE et Claudette TESTEMALE.
Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries.
Inf. Sci., 34(2):115–143, 1984.

[112] J. PROAKIS et D. MANOLAKIS.
Digital Signal Processing : Principles, Algorithms, and Applications.
Macmillan Publishing Company, New York, 1992.

[113] Gang QIAN, Qiang ZHU, Qiang XUE et Sakti PRAMANIK.
Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach.
ACM Transactions on Database Systems, 31(2):439–484, 2006.

[114] Kamisetty Ramamohan RAO et Patrick YIP.
Discrete Cosine Transform: Algorithms, Advantages, Applications.
Academic Press, Boston, 1990.

[115] Guillaume RASCHIA.
SAINTETIQ*: une approche floue pour la génération de résumés à partir de bases de données relationnelles*.

Thèse de doctorat, Université de Nantes, Décembre 2001.

[116] Guillaume RASCHIA et Noureddine MOUADDIB.
SAINTETIQ: a fuzzy set-based approach to database summarization.
Fuzzy Sets and Systems, 129:137–162, 2002.

[117] Dan RASMUSSEN et Ronald R. YAGER.
SummarySQL - a fuzzy tool for data mining.
Intelligent Data Analysis, 1:49–58, 1997.

[118] Rita A. RIBEIRO et Ana M. MOREIRA.
Fuzzy query interface for a business database.
Internationl Journal of Human-Computer Studies, 58:363–391, 2003.

[119] John T. ROBINSON.
The K-D-B-Tree: a search structure for large multidimensional dynamic indexes.
Dans Y. Edmund LIEN, réd., *Proceedings of ACM SIGMOD, Ann Arbor, Michigan*, pages 10–18. ACM Press, 1981.

[120] Daniel ROCACHER.
On fuzzy bags and their application to flexible querying.
Fuzzy Sets And Systems, 140:93–110, 2003.

[121] Alex ROUSSKOV et Duane WESSELS.
Cache digests.
Computer Networks and ISDN Systems, 30(22–23):2155–2168, 1998.
Disponible à l'adresse
citeseer.ist.psu.edu/rousskov98cache.html.

[122] Nick ROUSSOPOULOS et Daniel LEIFKER.
Direct spatial search on pictorial databases using packed R-Trees.
Dans *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 17–31. ACM Press, 1985.

[123] Alexander RUSSELL.
Necessary and sufficient condtions for collision-free hashing.
Journal of Cryptology, 8:87–100, 1995.

[124] David SADEK.
Le dialogue homme-machine: de l'ergonomie des interfaces à l'agent intelligent dialoguant.
Dans *Recueil des publications et communications externe du département RCP*, pages 215–261, Cnet, France Télécom, 1997.

[125] Régis SAINT-PAUL.
Une architecture pour le résumé en ligne de données relationnelles et ses applications.
Thèse de doctorat, Université de Nantes, Juillet 2005.

[126] Régis SAINT-PAUL, Guillaume RASCHIA et Noureddine MOUADDIB.
Mining a commercial banking data set: the SAINTETIQ approach.
Dans *Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'2002)*, Annecy, France, juillet 2002.

[127] Régis SAINT-PAUL, Guillaume RASCHIA et Noureddine MOUADDIB.
General purpose database summarization.
Dans *Proceedings of the 31st VLDB, Trondheim, Norvège*, pages 733–744, 2005.

[128] Michelle SCHATZMAN.
Numerical Analysis: A Mathematical Introduction.
Clarendon Press, Oxford, 2002.

[129] Bernhard SEEGER et Hans-Peter KRIEGEL.
The Buddy-Tree: an efficient and robust access method for spatial data base systems.
Dans *Proceedings of the 16th VLDB, Brisbane, Australia*, pages 590–601, août 1990.

[130] Timos K. SELLIS, Nick ROUSSOUPOULOS et Christos FALOUTSOS.
The R^+-tree: a dynamic index for multi-dimensional objects.
Dans *Proceedings of 13th VLDB, Brighton, England*, pages 507–518, 1987.

[131] Daniel R. SIMON.
Finding collisions on a one-way street: can secure hash functions be based on general assumptions?
Dans *Advances in Cryptology, Proceedings Euro-crypt'98*, volume 1403 de *LNCS*, pages 334–345. Springer-Verlag, 1998.

[132] Simon SINGH.
Histoire des codes secrets – de l'Egypte des Pharaons à l'ordinateur quantique.
Jean-Claude Lattès, Paris, 1999.

[133] Markku TAMMINEN.
The extendible cell method for closest point problems.
BIT, 22(1):27–41, 1982.

[134] Markku TAMMINEN.
Comments on quad- and octrees.
Communications of the ACM, 30(3):204–212, 1984.

[135] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP.
PostgreSQL 8.1.0 Documentation, 2005.
Disponible à l'adresse
http://www.postgresql.org/docs/.

[136] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP.
PostgreSQL 8.2 Developer Documentation, 2006.
Disponible à l'adresse
http://techdocs.postgresql.org/.

[137] Laurent UGHETTO, W. Amenel VOGLOZIN et Noureddine MOUADDIB.
Personalized database querying using data summaries.
Dans *Proceedings of the 15th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'06)*, pages 736–743, juin 2006.

[138] Laurent UGHETTO, W. Amenel VOGLOZIN et Noureddine MOUADDIB.
Résumés de données pour la personnalisation de requêtes.
Dans *LFA'2006*, pages 171–178, novembre 2006.

[139] Petrus Johannes Maria van OOSTEROM.
Reactive Data Structures for Geographic Information Systems.
Thèse de doctorat, University of Leiden, The Netherlands, décembre 1990.

[140] W. Amenel VOGLOZIN, Guillaume RASCHIA, Laurent UGHETTO et Noureddine MOUADDIB.
Interrogation de résumés de données et réparation de requêtes.
Dans *LFA'2004*, pages 239–246, Nantes, France, novembre 2004. Cépaduès Éditions.

[141] W. Amenel VOGLOZIN, Guillaume RASCHIA, Laurent UGHETTO et Noureddine MOUADDIB.
Querying the SAINTETIQ summaries – a first attempt.
Dans *Proceedings of the 6th International Conference on Flexible Query Answering Systems (FQAS2004), Lyon, France*, pages 404–417. Springer, juin 2004.

[142] W. Amenel VOGLOZIN, Guillaume RASCHIA, Laurent UGHETTO et Noureddine MOUADDIB.
Querying the SaintEtiQ summaries – dealing with null answers.
Dans *Proceedings of the 14th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'05), Reno, Nevada*, volume 1, pages 585–590, mai 2005.

[143] W. Amenel VOGLOZIN, Guillaume RASCHIA, Laurent UGHETTO et Noureddine MOUADDIB.

Querying a summary of database.
Journal of Intelligent Information Systems, 26:59–73, janvier 2006.

[144] Guoren WANG, Xiangmin ZHOU, Bin WANG, Baiyou QIAO et Donghong HAN.
A hyperplane based indexing technique for high-dimensional data.
Information Sciences, 177:2255–2268, 2007.

[145] Taehyung WANG et Phillip C.-Y. SHEU.
An object-oriented BSP Tree algorithm for hidden surface removal.
Int. J. Image Graphics, 2(3):395–412, 2002.

[146] Wei WANG, Jiong YANG et Richard R. MUNTZ.
PK-Tree: a spatial index structure for high dimensional point data.
Dans *FODO*, pages 27–36, 1998.

[147] Roger WEBER, Hans-Jörg SCHEK et Stephen BLOTT.
A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces.
Dans *Proceedings of 24th VLDB, New York City*, pages 194–205, août 1998.

[148] David A. WHITE et Ramesh JAIN.
Similarity indexing with the SS-Tree.
Dans *Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana*, pages 516–523, février 1996.

[149] Ronald R. YAGER.
On ordered weighted averaging aggregation operators in multicriteria decisionmaking.
IEEE Transactions on Sytems, Man and Cybernetics, 18:183–190, 1988.

[150] Ronald R. YAGER.
On linguistic summaries of data.
Dans *Knowledge Discovery in Databases*, pages 347–366. MIT Press, 1991.

[151] Lotfi A. ZADEH.
Fuzzy sets.
Information and Control, 8:338–353, 1965.

[152] Lotfi A. ZADEH.
The concept of a linguistic variable and its application to approximate reasoning.
Information Sciences, 8:199–249 & 301–357, 1975.
En trois parties, la troisième dans le volume 9, pages 43–80.

[153] Lotfi A. ZADEH.
International journal of applied mathematics and computer science.

Information and Control, 12:307–324, 2002.

[154] Ahmad ZANDI, James D. ALLEN, Edward L. SCHWARTZ et Martin BOLIEK.
CREW: Compression with Reversible Embedded Wavelets.
Dans *Data Compression Conference*, pages 212–221, 1995.
Disponible à l'adresse
citeseer.ist.psu.edu/166393.html.

[155] Justin ZOBEL, Alistair MOFFAT et Kotagiri RAMAMOHANARAO.
Guidelines for presentation and comparison of indexing techniques.
SIGMOD Records, 25:10–15, 1996.

Liste des tableaux

— *Corps du document* —

Liste des figures

— *Corps du document* —

Crédit photo

Page 66 : MoelSiabod.jpg @ Wikipedia Commons

http://commons.wikimedia.org/wiki/Image:MoelSiabod.jpg.

Table des matières

— Corps du document —

Le résumé linguistique de données structurées comme support pour l'interrogation

W. Amenel Abraham VOGLOZIN

Résumé

Le travail présenté dans cette thèse traite de l'utilisation des résumés de données dans l'interrogation. Dans le contexte des résumés linguistiques du modèle SaintEtiQ sur lequel se focalise cette thèse, un résumé est une description du contenu d'une table relationnelle. Grâce à la définition de variables linguistiques, il est possible d'utiliser des termes du langage pour caractériser les données structurées de la table. En outre, l'organisation des résumés en hiérarchie offre divers niveaux de granularité. Nous nous intéressons à fournir une application concrète aux résumés déjà construits. D'une part, nous étudions les possibilités d'utilisation des résumés dans une interrogation à but descriptif. L'objectif est de décrire entièrement des données dont certaines caractéristiques sont connues. Nous proposons une démarche de recherche de concepts et une instanciation de cette démarche. Ensuite, une étude des systèmes d'interrogation flexible, dont certains ont, ainsi que SaintEtiQ, la théorie des sous-ensembles flous comme base, nous permet d'enrichir la démarche proposée par des fonctionnalités plus avancées. D'autre part, nous avons intégré les résumés linguistiques de SaintEtiQ au SGBD PostgreSQL. L'objectif est d'aider le SGBD à identifier des enregistrements. Nous présentons un état de l'art des techniques d'indexation, ainsi que le détail de l'implémentation des résumés en tant que méthode d'accès dans PostgreSQL.

Mots-clés : SaintEtiQ, résumés linguistiques, données structurées, interrogation de résumés, sous-ensembles flous, interrogation flexible, aspects coopératifs, index, techniques d'indexation, méthode d'accès, PostgreSQL.

Linguistic summaries of structured data as a tool for querying

Abstract

This thesis deals with using summaries of data in a querying process. The work discussed focuses on the linguistic summaries of the SaintEtiQ model, in which a summary describes the content of a relational table. The definition of linguistic variables allows to use terms from the natural language to characterize the structured data. In addition, the organization of summaries into a hierarchy based on generalization links offers various levels of granularity. On one hand, we study the possible use of summaries in a descriptive querying process. The aim is to characterize entirely the data on the basis of a partial characterization. We propose an approach which consists in searching for specific concepts, from the expression of queries to the presentation of results. An instantiation of the approach using a summary hierarchy traversal algorithm is part of the proposal. Then, a survey of flexible querying systems, some of which are based on fuzzy sets as SaintEtiQ is, allows us to provide additional functionalities to the querying approach. On the other hand, we integrate linguistic summaries of the model into the DBMS PostgreSQL. The aim is to help the DBMS identify tuples by exploiting a summary hierarchy as an index structure. We provide a survey of indexing techniques as well as the details of implementing summaries as an index method under PostgreSQL.

Keywords: SaintEtiQ, linguistic summaries, structured data, querying of summaries, fuzzy sets, flexible querying, cooperative aspects, index, indexing techniques, access method, PostgreSQL.

Discipline : INFORMATIQUE
Spécialité : INFORMATIQUE

Laboratoire : LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.